A visual BIM query language

CONFERENCE PAPER · SEPTEMBER 2014

DOI: 10.13140/2.1.2158.9769

3 AUTHORS, INCLUDING:

Alexander Wülfing

Hochschule für Technik und Wirtschaft Dre…

4 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Ronny Windisch

Iproplan, Consulting Engineers & Architects

9 PUBLICATIONS **14** CITATIONS

SEE PROFILE

A visual BIM query language

A. Wülfing, R. Windisch & R. J. Scherer
Institute of Construction Informatics, Technische Universität Dresden, Germany

ABSTRACT: The change from drawing-based to model-based work demands the integration of the very important group of the craftsmen into the BIM-oriented process chain, which is practically not considered in BIM development until now. Effective and encompassing access to the particular information needed for a certain work task based on different domain specific models which are the fundament for a 5D-BIM (building, costs and time) with respect to the specific working environment is a major prerequisite to achieve craftsmen integration. Because it should not be necessary to burden craftsmen with learning some kind of (text based) programming or query languages to formulate model queries this paper discusses the concept and the language elements which will be needed for a Visual BIM Query Language. Such a visual language is supposed to lower the threshold and effort required for the adoption of the BIM-based work paradigm by the small and medium crafts enterprises.

1 INTRODUCTION

Construction works require an immense number of project partners respectively involved building trades. Traditional cooperation between these trades respectively the involved craftsmen and tasks like the determination of quantities, creation of parts lists, work scheduling etc. are in most cases done by hand based on traditional information resources (i.e. semi- or unstructured documents like construction drawings, specifications for building systems etc.) especially in the surrounding of small and medium-sized companies till this day. Efficient search and retrieval of information which are needed for a particular work task can hardly be performed and require additional effort for manual rework since the relevant information is distributed in different documents that have to be searched trough and integrated manually. Also the change in used (proprietary) data formats (e.g. for floor plans and construction drawings, bill of quantities and work plans) and the integration of these different information resources causes different problems. On the one hand additional synchronization problems occur if information which is not stored centralized is changed. On the other hand required conversion between different formats is a problem as well. These tasks and problems are solved mostly by hand which is very time consuming and error-prone.

The BIM concept based on the fact that as much as possible building information generated during the lifecycle of a construction work is stored in a commonly shared information model avoids the change and transformation in different media formats and provides for an encompassing unified information resource. But of course not all of the information stored in a BIM is relevant for craftsman's work and/or not all needed information could be integrated into only one single model (separation of concerns). Consequently the search and retrieval of needed information is not a trivial task in such huge and domain comprehensive models which possibly are (implicitly) linked with other BIM-related information resources like cost and time models. Therefore appropriate filter and query methods and tools are required which enable the craftsmen companies to access BIM-based information resources with respect to their specific needs and working environment.

The presented work is part of the German R&D project eWorkBau (www.ework-bau.de) that is concerned with the integration of craftsmen into the BIM process with regard to problems like information extraction from a 5D-BIM and the mediation of basic skills to deal with the special problems and needs which will arise in the context of a BIM based project environment during the bidding and construction phase in contrast to the present working method. In the paper we discuss key aspects of information extraction using the proposed visual BIM query language (VBQL) especially with focus on the language elements needed to specify filter queries in

a way that is suitable for craftsmen of different trades.

1.1 *Craftsmen's place in the BIM process*

The integration of craftsmen into the BIM process chain depends on the considered working scenario. In context of the eWorkBau project new construction and cleansing will be the starting point for further considerations relating to the work with BIM. In both scenarios the craftsmen need to use filter tools to retrieve the particular information subsets relevant to their specific work tasks. For both scenarios the craftsman will use some existing computer processible BIM models handed over by the building owner or the client respectively at the end of design phase (e.g. 3D CAD BIM). This includes not only the usage of building models but the usage of construction schedules and bill of quantities as well. Construction schedules will be used e.g. for the scheduling of the working hours of employees, scheduling and coordination of material orders, resource scheduling regarding tools and equipment or the coordination with other trades. In order to carry out the aforementioned tasks the craftsman will need in most cases the bill of quantities and the building model as well. In the role of a (potential) contractor the craftsman will use the information contained in a 5D-BIM to create an offer related to a particular construction work. The information needs depending on the craftsman's trade (e.g. plumbing, tiling, electrics) and the particular task at hand in each case.

1.2 *Why visual query languages?*

For the retrieval of craft specific information from a (5D-) BIM it is necessary that a craftsman, as end user of filter software, can express filter queries like "Select all windows in the 5th floor having no sunblind" or "Get the number of electrical power outlets in all office rooms of 5th floor" in an easy and trade specific way. The filter queries are processed by a corresponding filter-software which has to be flexible enough to adapt on various filter requirements. The craftsmen should not have to learn some kind of programming languages like Java or other text based query languages to formulate model queries. In particular an appropriate filter method has to enable a separation between the model semantics defined by the underlying data model(s) and the semantics of the domain specific concepts used by the craftsmen. This separation allows the craftsmen the usage of well-known domain concepts for the specification of query expressions without knowing the concepts and structure of the underlying data model(s). Therefore a graphical solution/visual language which allows for graphically supported specification of domain specific filter queries is preferred. Such a language is easier to learn and use (no programming skills are needed) and provides for direct visual feedback that supports the minimization or elimination of mistakes in the creation of filter queries by the user if the graphical representation of the language elements is designed properly e.g. if only valid arguments can be assigned to a filter specification. Additionally, a well designed graphical representation of the language elements may help to guide the user trough the filter specification process in an intuitive way. This enables the user to concentrate on his particular work task rather than on language structure and syntax.

1.3 *Filter and query languages for BIM*

Filter and query languages for BIM are mostly dedicated to the use with IFC-based model data since the IFC project model (www.buildingsmart-tech.org/ifc) is been widely accepted in the AEC industry and is broadly used for information exchange and collaboration between the different actors involved in a building project. Due to the high complexity and size of the IFC project model and the heterogeneous information requirements of the different disciplines and actors involved in the building life cycle various filter and query languages have been developed. The different approaches can be divided into generic or meta level languages, domain specific languages and languages respectively approaches dedicated to a particular field of problem. Examples for the first type of languages are the EXPRESS Query Language (EQL, Huang 1999) as well as the Partial Model Query Language (PMQL, Adachi 2002) designed for querying EXPRESS-based model data. Since IFC is based on the EXPRESS modelling language (ISO 10303-11) both can be also used to query IFC model data. Whereas EQL can be used to query IFC in file format (i.e. STEP physical file format, ISO 10303-21) PMQL provides for standard select, update and delete operations on IFC model data stored in an IFC model server. An approach related to the second type of languages is the BIM Query Language (BIMql) that is especially dedicated to querying IFC model data following the CRUD (create, read, update and delete) principle (Mazairac & Beetz 2013). BIMql aims to enable the use of domain specific concepts in query expressions instead of complex descriptions of the relevant IFC model structure describing the requested subset of model data. This requires additional programming logic in order to specify how the domain specific concepts can be derived from the IFC data model.

The third type of languages is quite similar to domain specific query languages but is dedicated to a more restricted field of problems. The Spatial Query Language is an example of such an highly specialized approach since it addresses advanced geometrical and topological querying capabilities based on spatial reasoning and topological analysis

operations (Borrmann & Rank 2009). Another problem specific query language is the Built Environment and Rule Analysis language (BERA, Lee 2011) that is specifically designed for the support of technical BIM analysis and validation tasks which are processed in the architecture domain.

However, almost all of the aforementioned languages are designed as text based languages and require either programming knowledge or knowledge about the underlying data models (e.g. IFC) or both to some extent. Additionally, the envisaged application areas are mainly oriented on the needs of the design phase and consequently taking no other domain models (like cost or time model) into account and are more suited for the use by domain experts like architects, structural or mechanical engineers.

2 VISUAL LANGUAGES

The vocabulary (i.e. the set of words) and the rules which specify how to define valid expressions (syntax) are elementary parts of a (formal) language. In contrast to one-dimensional string languages, where expressions are composed of textual symbols (Costagliola et al. 1997) the vocabulary of a visual language consists of graphical objects, also called visual elements or graphical symbols, which can be composed to visual expressions, also called pictures or diagrams. The composition of visual expressions is based on rules which define if a visual expression belongs to a visual language (Costagliola et al. 1997, Costagliola et al. 2002, El Kouhen et al. 2013, Costagliola et al. 1998). Although many visual languages don't make such a distinction (Fondament & Baar 2005) the syntax of a (visual) language can be further divided into an abstract and a concrete syntax. The abstract syntax defines the structure of the language, i.e. the language concepts, whereas the concrete syntax deals with the actual representation of the language elements presented to the user. This classification allows for multiple representations for the same structure if a proper mapping is available which maps the language elements to a concrete graphical representation (Figure 1). Besides the definition of the language concepts, the description of the language semantics, i.e. the meaning which is assigned to the language construct, is another vital point in the definition of a visual language (Fondament & Baar 2008, Tveit 2008).

Because the complexity of text based programming languages makes the access for programming beginners difficult, visual (programming) languages like Scratch (Maloney et al. 2010) are used in the surrounding of teaching basic concepts of computer programming. So the user can concentrate on problem solving and must not think about language syntax. Another prominent example for a visual language is UML (Unified Modeling Language), which is used as modeling language in design and modeling of (object-oriented) software-systems specified by the Object Management Group (OMG 2010).

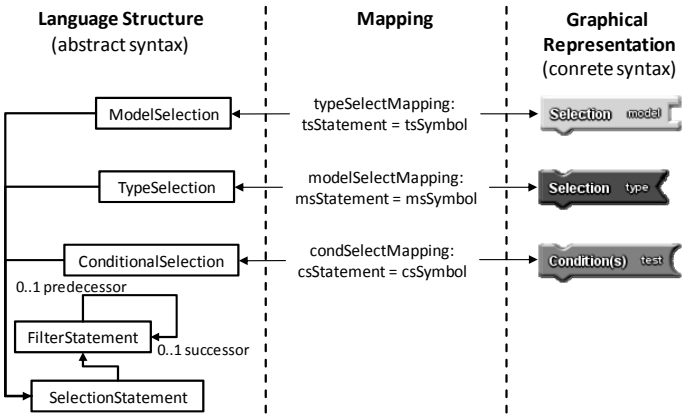

Figure 1: Relationship between language structure (abstract syntax) and language representation (concrete syntax)

3 VISUAL BIM QUERY LANGUAGE FOR CRAFTSMEN

3.1 *Requirements for a VBQL*

The requirements for a Visual 5D BIM Query Language (VBQL) arise both from the use of multiple domain models (building model, cost model, time model) as well as from the special needs of craftsmen with respect to such language. The information extraction using current available tools demands either knowledge about the underlying data structure of the model to retrieve the information, is time consuming and error-prone (e.g. by removing unwanted elements in a viewer manually) or can't be done because the needed information is only implicitly described in the data model(s). For this reason the language elements in a visual BIM query language should be reusable as far as possible for all trades but should also respect domain specific elements. Therefore in a first step the information needs for the different crafts especially for the tendering and construction phase were determined in co-operation with the craftsmen involved in the project. It turned out that the support for the creation of parts lists and the retrieval of assigned information like material, position and type are important points for the craftsmen across the trades in these phases. For example for an electrician the number and type of switches, lighting parts and electrical outlets are more interesting than the number of slabs. Based on these information needs the required general language elements can be classified in selection statements, information extraction statements and additional operators. Selection statements select elements from an object set according to a selection criterion which can be regarded as an argument of the selection statement. Information extraction statements are needed to specify which information

from the selected objects should be retrieved (e.g. assigned material). Additional operators are needed to determine information which is either implicit or explicit represented in the underlying data model(s). An example for implicit information would be the calculation of dimensions and quantities based on the geometry of an object, e.g. the length of an electric cable segment or the roof area. An example for explicit information could be assigned material information that is accessible by evaluating explicit object relations.

3.2 *Language elements of the VBQL*

3.2.1 *Filter Query*

A filter query can be regarded as a sequence of filter statements. A filter statement has zero or one predecessor and zero or one successor which is a filter statement as well. Each filter statement has either zero or more primitive or composed arguments or the output which is delivered by one or more other statements is used as argument for the statement itself. A primitive argument is an argument which is a single value from a certain range of values (e.g. a number). A composed argument is an argument which itself has one or more arguments (either primitive or composed). Each valid filter query is been represented by an ordered set of instantiated visual language elements composed according to the syntax rules of the VBQL. The elements as part of the language structure (described here) are related to the visual elements by a mapping between the abstract and the concrete syntax.

3.2.2 *Filter Statements*

A filter statement is the abstract supertype of all statements. It operates on a set of values if an input is given and returns a set of values with one or multiple elements which can be passed to another statement or displayed to the user. A filter statement can be either a selection statement or an information extraction statement. Each valid filter query is composed of one (model) selection statement and multiple optional selection or information extraction statements.

3.2.3 *Selection Statements*

- *Model selection*

The user of the VBQL should be aware on which domain model he will operate. So the (primitive) argument of a model selection specifies a particular domain model and thereby a set of objects on which a filter query operates. Therefore a selection by model statement has to be the first statement in each valid filter query.

- *Type selection*

The type selection can be contained optionally in a filter query and follows after a model selection statement if it is applied. The type selection takes as argument one or more object types which have to be given by the user and must be contained in any filter query. A set of all objects which are of the given type(s) will be returned. A selection by object type is relevant for all domain specific models. As already mentioned one important task for the craftsmen is the creation of parts lists for which a type selection will be useful amongst others. The passed type argument should consider the concept of generalization and specialization as well. For example walls, columns, slabs etc. can be generalized to the type building element which allows for querying all building elements contained in a given building model. An example for a specialization would be the distinction of elements of type electrical switch in intermediate switch and toggle switch.

For the time model, which for example contains dates for the work schedule of the craftsmen, the type selection could also be useful. Because a date can be considered as an element of the type “date” it is possible to query all dates. The set of all dates alone is not very useful but is the basis for further restrictions, e.g. dates in a certain time span, in combination with other language elements (see time conditions). For the bill of quantities, which contains construction work elements which are assigned with costs, such a type selection could be used for the selection of all construction work elements of a particular type.

- *Conditional selection*

A conditional selection selects objects from an object set based on one or more conditions. All objects which fulfill this condition(s) will be returned as output of the selection statement. In contrast to the type selection the argument of a conditional selection is not an argument which relates to a certain object type but a condition argument. Of course a type selection could be regarded as a conditional selection like “is of type”. But this distinction enables an intuitive top-down creation of filter queries by the user starting with model and type selection to restrict the filter result in a first step. This is also an important point for a later syntax checking.

A conditional argument represents a condition which evaluates either to true or false for elements of the conditional selection input. The argument of a selection by condition can be of the following types:

- spatial condition,
- time condition,
- comparison condition,
- logical condition and
- relational condition.

A **spatial condition** can be applied to restrict results of filter statements which operate on the building model. Examples of spatial conditions would be “in storey”, “in space”, “in building” or “in zone” which could be used to determine all (building) ele-

ments in a certain storey, space, building or spatial zone. So the spatial condition deals with the spatial containment of elements in a spatial zone.

A **time condition** can be applied to restrict results of filter statements which operate on the time model. The most important time condition is "time period" which delivers all dates in a certain time period. Other conditions are "in calendar week" (all dates in a certain calendar week), "at calendar date" (all dates at a certain calendar date) and "before" and "after". The "in calendar week" and "at calendar date" conditions can be derived from "time period". In case of "at calendar date" the start date and finish date of the time period would be equal and in case of the "in calendar week" condition the start date of the time period would be the start of the specified calendar week and the finish date would be the finish date of the calendar week. The "before" and "after" condition, which has as argument a time period or a single date, can be used to retrieve all dates before or after a certain date or time span. This could be useful for example to check if employees, tools or machines are scheduled before or after a certain date or not.

A **comparison condition** takes two arguments as input and compares these arguments based on the well-known relational operators from programming languages like less than (or equal), greater than (or equal), equal and unequal. The argument of a comparison condition can be either a number or an operator which returns a number.

AND, OR and NOT are logical operators which can be used to formulate **logical conditions** and to create more complex conditions like "select all spaces which are NOT in storey one". Logical operators (AND, OR, NOT) can also be used to apply multiple object types and conditions in a type and conditional selection like "select all objects of type space which are in storey one AND where the space area is smaller than 15 square meter" or "select all objects of type wall AND column where height is smaller than 3.50 meter AND which are in storey one".

Relational conditions deal with structural, functional or operational relationships between elements, e.g. to select all windows placed in a given wall object or to select the particular wall object where a given set of electrical power outlets is placed in or has to be installed respectively. The evaluation of according relational conditions involves additional operations in case the used relational predicate (e.g. "connected_to", "placed_in" or "supplied_by") is not been represented explicitly in the underlying data model (e.g. IFC).

3.2.4 *Information Extraction Statements*

Information extraction statements are used to specify which information a user wants to retrieve from selected objects by using additional operators (see section 3.2.5). Information to be extracted for a selected set of objects may be for example the assigned material or geometrical information like position, surface area or volume. The application of operators to specify the information requested by the user enables the separation between the domain specific concepts used in the VBQL and the concepts of the underlying data models. Therefore it can be seen as a mapping between the concepts of the particular craftsman application model and the relating information represented either explicitly or implicitly in the underlying data models.

An information extraction statement can be either of **standard** or of **extended** type. Standard information extraction statements are predefined and can't be customized by the user. An example for such a standard information extraction statement could be "getAssignedMaterial" which retrieves the material information each assigned to the elements of a selected object set. Another example would be a statement which retrieves information according to a standard (pre-configured) room data sheet or a parts list, e.g. containing a set of selected building elements, related spatial containment and additional geometrical information. An extended information extraction statement offers more flexibility concerning at least two points. It can be specified where and in which form the information should be displayed or stored (e.g. on screen or in a file). Additionally, it is possible to specify precisely which information should be retrieved. In the context of eWorkBau examples for extended information extraction statements are customized parts lists or room datasheets where the user can specify in detail which information should be contained and if the information is displayed to the user or exported into an (XLS) file.

3.2.5 *Additional Operators*

The set of additional VBQL operators extends the set of basic operators (comparison and logical operators) and can be divided into aggregation operators, set operators and operators for determination of implicit and explicit information. Operators may be used in selection statements (e.g. as a predicate or a condition argument) and extraction statements as well. A single operator encapsulates a particular piece of operational logic and can be mapped to a language element consisting of a graphical shape and a domain specific keyword indicating the assigned operation or the type of value(s) returned by the operator respectively.

- *Aggregation Operators*

Similar to the definition of aggregation functions in database systems an aggregation statement returns a single number value based on multiple values as input (e.g. object count).

Because the creation of parts lists respectively the determination of element quantities is an important

demand for the craftsmen it should be possible to **count** elements. In combination with the selection by type it would be for example possible to determine the number of windows contained in a building model. So the input of the count operator is a set of objects or a set of attribute values and the output is the number of elements in the set.

The **min/max** operators can be used to determine the smallest/largest element in a set based on a comparison respectively order relation. This can be used for example to determine the smallest/largest area(s) for all spaces in a building if the input is a set (of numbers) which contains all areas off all spaces. Because an (order) relation is not limited to compare numbers a min/max operator could be used to determine the room with the smallest/largest area if the input is a set of spaces which can be compared relating to its areas.

Additional aggregation operators which will be supported by VBQL are the **sum** and the **avg** operators. The sum operator enables e.g. the calculation of the total sum of all space areas whereas the avg operator can be used e.g. to calculate the average area of all spaces located in a building.

- *Set operators*

Set operators operate on (object or attribute value) sets and the result is another set.

Taken two or more sets as input the **union** U returns a set which contains all elements which are contained in set A and/or set B and/or set C and so forth.

So the union U for the sets A, B, C, … is $U = A \cup B \cup C \ldots = \{ a \mid a \in A \vee a \in B \vee \ldots \}$

A union could be used for example to use another criteria or values in a conditional selection to restrict a type selection. For example if a user wants all walls in storey one and all doors in storey two he could use a type selection “select all walls” and a conditional selection “in storey one” to get all walls in storey one and a type selection “select all doors” and a conditional selection “in storey two” to get all doors in storey two. The union of these two sets contains all walls in storey one and all doors in storey two.

Taken two or more sets as input the **intersection** returns a set which contains all elements which are contained in all involved sets. For the craftsmen such an operation could be useful, in context with the time model, to check if there are date conflicts (e.g. if the work scheduling conflicts with holidays).

Taken two sets A and B as input the **difference** (A\B) is the set that contains all elements which are contained in A but not in B. An example in context of the building model would be the retrieval of all columns in the 5th storey which has not a certain material. One query would select all columns in the 5th storey and another query would select all columns in the 5th storey with a certain material. The intersection of these results would contain all columns in the 5th storey with another material. Of course this result could be also achieved by using a conditional selection like “NOT assigned material”.

- *Operations for implicit information*

Some information which a craftsman wants to retrieve respectively use in a filter specification is being represented only implicitly in the (building) model but can be derived from other model data. This includes especially geometrical information, like distances, heights, areas etc., which has to be derived from the geometrical representation of the elements. But also topological concepts like above, under or beside as well as information related to relational conditions or predicates, e.g. the evaluation of the “placed_in” predicate to identify the electrical components placed in a particular wall object.

- *Operations for explicit information*

Also information which is explicitly contained in a model, for example through analysis of object references, demands knowledge about the underlying data model(s). To make this for a user transparent the visual query language must provide elements which can be used to gain such information. This includes the access e.g. to assigned materials or property sets which contain additional information requested by the user.

4 IMPLEMENTATION AND USE CASE EXAMPLES

4.1 *BIMcraft prototype*

After the determination of the information needs for the different crafts in the tendering and construction phase in co-operation with the craftsmen and based on the outlined concept of a VBQL a first (Java) based software prototype (BIMcraft) was developed. BIMcraft allows for specification of filter queries using some of the above mentioned language elements. For the access on IFC based building models the filter toolbox BIMfit (Wülfing et al. 2012) is used. BIMfit is a JAVA-based filter toolbox for generating subsets of an IFC-based building model that fulfill certain domain or task specific information requirements. It contains a set of hierarchically ordered basic filter functions which can be dynamically assembled to higher order filter operations with regard to the given application context at hand. The BIMfit filter method is based on a breakdown of single application specific filter operations into several reusable and configurable filter functions encapsulating a particular piece of functional logic. Each filter function is assigned to one of three different levels of abstraction which together establish the Neutral Layer of the underlying generic filter

framework (Windisch et al. 2012). Each layer implements the operational mapping to the concepts of the upper layer since each function is specified by using functions of the layers below. This approach allows for providing filter functionality tailored for a considerable amount of different application contexts based on a finite set of pre-defined filter functions implemented in the BIMfit toolbox.

For the implementation of the visual language respectively visualization of the VBQL language elements the java library OpenBlocks (http://education.mit.edu/openblocks) is used which supports the development of a visual (block programming) language amongst others by allowing the specification of language elements and the graphical interface represented to the user based on a description in an XML file and providing of an event-management (Roque 2007). The framework allows for specification of the language structure (i.e. language elements and valid element relations) and maps these language elements to a predefined set of visual elements (i.e. symbols).

4.2 *Examples*

In the following several examples are shown which are already implemented in the filter software prototype BIMcraft. Figures 2 and 3 present two filter queries each consisting of a model selection as first statement and a type selection as second statement. The model selection specifies that the type selection will operate on the underlying building model. The type selection in the first filter query (Figure 2) has only one passed argument ("Space") which is connected to the type selection. The second connector is blank. Additional types can be attached if needed. This is the case in the second filter query (Figure 3) which selects all walls and doors i.e. objects of type *IfcWindow* and *IfcDoor*. Depending on the amount of attached type arguments the type selection symbol will expand dynamically as you can see.

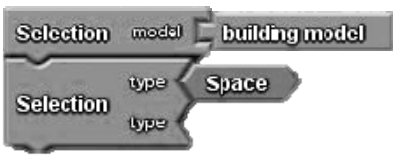

Figure 2: Type selection for type "space" which is applied on the given building model.

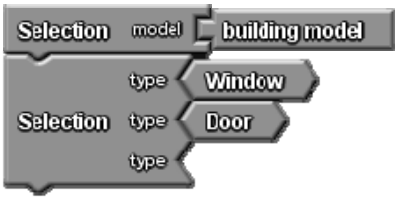

Figure 3: Type selection for type "window" and "door" which is applied on the given building model.

Figure 4 shows a more complex filter query example. This query consists of a model and a type selection in combination with a conditional selection which restricts the set of all spaces. The conditional selection has two condition arguments. The first argument specifies that only spaces in the first storey should be regarded (spatial condition). The second argument specifies the area of the spaces which must be smaller than 13 square meters (comparison condition). The space area is retrieved by using the "area"-operator which calculates the ground area based on the geometrical representation of *IfcSpace* objects. Hence the query shown in Figure 4 filters all spaces from the building model which are contained in storey one and have an area smaller than 13 square meter.

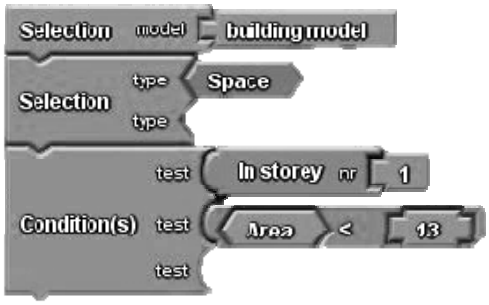

Figure 4: Filter query combining type and conditional selection which operates on the given building model.

Figure 5 shows the current GUI of the BIMcraft prototype including the editor for the graphical filter specification and a 3D model viewer (Open IFC Tools, www.openifctools.org) for the visualization of the filter query result. In the example, shown in Figure 5, all windows and doors are selected (and therefore showed in the viewer). Additionally the standard information extraction statement "Assigned material(s)" is used to specify the information to be retrieved for the selected wall and door elements and displayed to the user (not shown).

5 CONCLUSIONS

This paper discussed the concept and design of a visual BIM query language (VBQL), especially tailored for the needs of craftsmen in the context of a BIM-based project environment. Such a language is an important part to use and retrieve information from a 5D-BIM which can't be retrieved until now because of lack of knowledge about the underlying data structures and the derivation of implicit and explicit information or even if this knowledge is available because of the time consuming extraction of needed information (e.g. by using just a model viewer). The possible language elements of the VBQL allowing for efficient, intuitive and easy to learn and use specification of filter queries and the

prototype BIMcraft implementing the designed VBQL concept were presented. Although the developed prototype provides not all of the discussed language elements at present it can be already used to show some advantages concerning information extraction in contrast to (semi) manual work which is still dominant in the daily work of craftsmen.

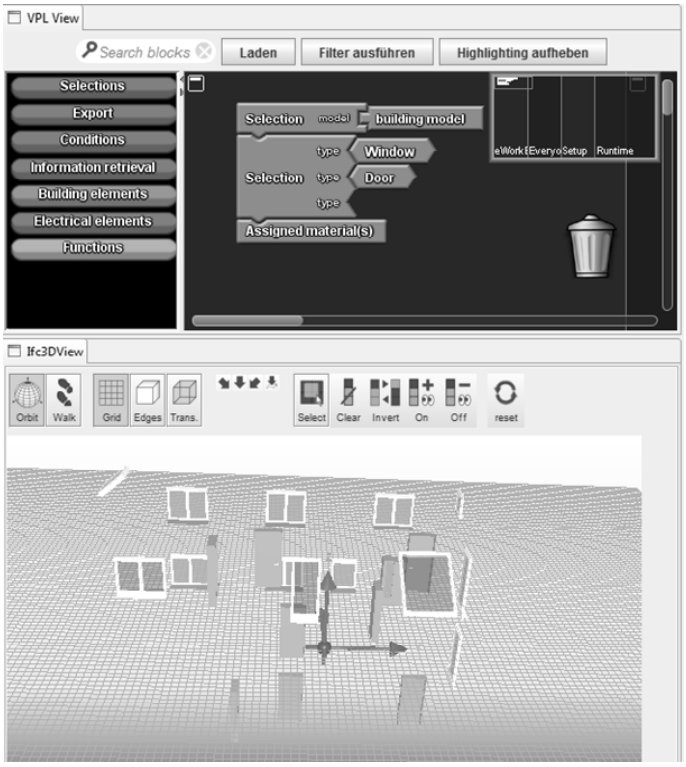

Figure 5: BIMcraft GUI – upper left part: library which contains the available query elements; upper right part: query editor; lower part: model and query result visualization

ACKNOWLEDGMENT

The presented work is performed in the frames of the ongoing research project “eWorkBau” (Grant-No. 01PF07045A, www.ework-bau.de) with financial support of the German Federal Ministry of Education and Research and the European Social Fund. This support is herewith gratefully acknowledged.

REFERENCES

Adachi, Y. 2002. Overview of Partial Model Query Language. VTT Report VTT-TEC-ADA-12, Espoo.

Borrmann, A. & Rank, E. 2009. Topological analysis of 3D building models using a spatial query language. Advanced Engineering Informatics, DOI:10.1016/j.aei.2009.06.001.

Costagliola, G., De Lucia, A., Orefice, S., Tortora G. 1997. A Parsing Methodology for the Implementation of Visual Systems. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. 23, No. 12, December 1997, pp. 777-799.

Costagliola, G., De Lucia, A., Orefice, S., Tortora, G. 1998. Positional Grammars: A Formalism for LR-Like Parsing of Visual Languages. In: K. Marriott, B. Meyer (Eds.), Visual Language Theory, Springer, pp. 171–192.

Costagliola, G., De Lucia, A., Orefice, S., Polese, G. 2002. A Classification Framework to Support the Design of Visual Languages. In: Journal of Visual Languages & Computing, Vol. 13, Issue 6, December 2002, pp. 573-600.

El Kouhen, A. & Gérard, S. 2013. A Component-Based Approach for Specifying Reusable Visual Languages. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), September 2013, pp. 135-138.

Fondement, T. & Baar, T. 2005. Making Metamodels Aware of Concrete Syntax. In: Hartman, A., Kreische D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, Springer, pp. 190-204.

Huang, L. 1999. EXPRESS Query Language and Templates and Rules: Two languages for advanced Software System Integrations. Dissertation, Ohio University.

Lee, J. K. 2011. Building Environment Rule and Analysis (BERA) Language - And its Application for Evaluating Building Circulation and Spatial Program, Dissertation, College of Architecture, Georgia Institute of Technology.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. 2010. The Scratch Programming Language and Environment. In: ACM Transactions on Computing Education (TOCE), Volume 10, Issue 4, November 2010, pp. 16:1-16:5.

Mazairac, W. & Beetz, J. 2013. BIMQL – An open query language for building information models. In: Advanced Engineering Informatics, Volume 27, Issue 4, October 2013, pp. 444-456.

OMG 2010. OMG Unified Modeling Language (OMG UML), Infrastructure. Version 2.4.1. Online available at: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF

Roque, R. V. 2007. OpenBlocks: An Extendable Framework for Graphical Block Programming Systems. Masterthesis. Massachusetts Institute of Technology. http://hdl.handle.net/1721.1/41550

Tveit, M. S. 2008. Specification of Graphical Representations – using hypergraphs or meta-models?. In: Norsk informatikkonferanse (NIK), November 2008, pp. 39-50.

Windisch, R., Wülfing, A., Scherer, R. J. 2012. A Generic Filter Concept for the Generation of BIM-based Domain- and System-oriented Model Views. In: 9th European Conference on Product and Process Modelling (ECPPM), Reykjavik, Iceland, 25-27 July 2012, pp. 311-319.

Wülfing, A., Baumgärtel, K., Windisch R. 2012: BIMfit – A Modular Software Tool for Querying and Filtering of Building Models (in German). In: Proc. 24th Conference “Forum Bauinformatik”, Bochum, Germany, 26-28 September 2012, pp. 9-16.