Towards a Framework for a Domain Specific Open Query Language for Building Information Models

Wiet Mazairac, Jakob Beetz
Eindhoven University of Technology, The Netherlands
l.a.j.mazairac@student.tue.nl

Abstract. In this paper we present the on-going development of a framework for a Domain Specific Open Query Language for Building Information Models. This query language will make it possible to retrieve data from building information models stored on the open source bimserver.org model server. Even though some partial solutions to this problem already have been suggested, none of them are open source, domain specific, platform independent and implemented at the same time. This paper provides an overview of existing approaches and conceptual sketches of the language in development.

1. Introduction

Being able to obtain required information in time is one of the keys to success in the building industry. Not too long ago, drawings were stored using file cabinets and sent by post. An index simplified the process of retrieving the drawing needed. Over time the building process has become more complex. The number of stakeholders involved in a design and construction process has increased and so did the amount of information each actor generates. More recent approaches to share and distribute information include shared building information models stored on specialized servers, which enable the structured maintenance of large quantities of data from various sources. New or altered designs can be uploaded to the server and only a few moments later those design changes are at the disposal of others.

For an average building and construction project, the amount of information stored in a multi-domain repository becomes very large and complex. However, information needs of various project stakeholders differ substantially. For example, a construction engineer does not need all the information stored in the common model and is e.g. not interested in the type of suspended ceiling used in a building design. This makes the extraction of partial model subsets or domain specific views on large models necessary. On a generic level, this is addressed by the Information Delivery Manual (IDM) effort, and the Model View Definitions (MVD) that are under ongoing development. However, these approaches rely on fixed sub sets of information and do not allow an easy, project-specific assembly of views on the fly. Although various approaches have been proposed for selecting and filtering data from a server on which a building information model is stored, an open source, platform independent solution for creating such queries is currently not available.

Most software environments designed to process building information models provide some way to select or filter data. The BimServer.org project (Beetz et al, 2010) for example enables end-users to download parts of the model from the server by providing GUIDs, selecting all instances of a particular class, using filters (e.g. as generated by the mvdXML tool by Weise et al) or by writing custom queries in JAVA. Other tools such as the Solibri Model Viewer (http://www.solibri.com/) also provide ways to select part of the model. Although it offers partial model extraction, sophisticated queries and constraint checks, these mechanisms are not based on open, reusable specifications and cannot be tailored to individual needs in straight-forward, non-proprietary ways.

2. Overview and analysis of existing querying approaches

Already, a number of querying approaches are available in order to extract data from large model instances. They can be divided into two groups. The first category includes generic querying approaches. These are more versatile, but might not be able to perform a very specific task. Approaches specific to the AEC/FM domain fall into the second category. In this section we provide an overview of these two categories of BIM querying approaches.

2.1 Generic Querying Approaches

Many of the existing database applications on the market use the Structured Query Language (SQL) as the standard language. SQL was designed for managing data in a Relational Database Management System (RDBMS). SQL makes it possible to create, read, update and delete (CRUD) records. Designed on top of the .NET platform, the Language Integrated Query (LINQ) is Microsoft's technology to provide a language-level support mechanism for querying data of all types. These types include in-memory arrays and collections, databases, XML documents, and more. The Resource Description Framework (RDF) is a directed, labeled graph data format for representing information in the Web. RDF is often used to represent, among other things, personal information, social networks, metadata about digital artifacts, as well as to provide a means of integration for disparate sources of information. Although a number of different RDF query languages exist, the SPARQL Protocol and RDF Query Language (SPARQL) is the most popular one and has been officially standardized by the W3C. The Object Constraint Language (OCL) is a language for precise textual descriptions of constraints which apply to graphical models captured in the Unified Modeling Language (UML).

2.2 BIM querying Approaches

Large and complex engineering models have spurred the need for the creation of query languages already over a decade ago. One of the early examples is the Express Query Language (EQL) proposed by Huang (1999) designed as a generic query extension to STEP initiative. The Partial Model Query Language proposed by Adachi (2002) aims to provide a general means for select, update, and delete partial model data that contains specific part of product model data. This language enables users to write recursive and conditional expressions based on SQL. However it does not provide the possibility to create or add model data to an existing building information model. The Georgia Tech Process to Product Modeling by G. Lee et al, (2006) is a product modeling method to (semi-) automatically drive a product model from collected process information. A process modeling module (called the Requirements Collection and Modeling (RCM) module) can capture the contents, scope, granularity, and semantics of information used in a process model. Later, the captured information can be structured as a product model. GTPPM does not support several IDM implementation details, it cannot automate the generation of an entire IDM. Benefits of using GTPPM as a method to create an IFC IDM view include traceability and reusability. The Generalized Model Subset Definition (GMSD) schema devised by Weise et al (2003) enables the realization of client/server or file based transactions in a structured manner, at different levels of granularity, and for different data exchange formats. GMSD is specifically oriented to the support of EXPRESS-based models, with special attention to IFC. GMSD is not a language per se but a schema which allows a neutral definition format with possible mappings for various practical data exchange and server/client realizations. Borrmann et al (2006) introduced the concept of a spatial query language for Building Information Models. It provides formal definitions using point set theory and point set topology for 3D spatial data

types as well as the directional, topological, metric and Boolean operators employed with these types. It also serves to outline the implementation of 3D spatial query processing based on an object-relational database management system. The commercial application “Solibri Model Viewer” provides several ways to select or view parts of the Building Information Model. However the methods of selection and filtering apply to this software package only. The selection and filtering methods are not platform independent and therefore cannot be exported to or imported from other software packages.

3. Design of a query language for building information models

The framework to be developed should enable the end user to interact with a building information model following the CRUD principle. By using this framework, records in the building information model can be created, selected, updated, and deleted. Adhering to the network-like structure the IFC model exposes in particular ways, recursive queries should be made possible. This enables the end user to collect information even from objects that are related to other objects via an arbitrary number of relationships edges. By using a syntax close to the natural language non-expert users should be enabled to use these querying mechanisms on an ad-hoc, per-project level. Instead of using the lexically correct IFC class names such as IfcWallStandardCase and IfcDoor, we would like to enable users to use natural language terms such as ‘Wall’, ‘Walls’ and ‘Door’ in the end. Using look-up dictionaries and structured vocabularies such as the International Framework for Dictionaries (IFD), localizations of the query language can be achieved.

Objects in a building information model often are linked to each other through a complex network of relationships. A window for example is related to an opening, that opening is related to a wall and the wall is related to a building storey etc. Often, even seemingly straight-forward relations between information entities in a BIM require complex navigation of the underlying model. To retrieve a simple property such as the fire rating of a given door for example demands already quite some knowledge of the underlying data model from an interested domain expert and results in a complex query: The desired value is not directly provided as an explicit attribute of e.g. the IfcDoor entity, but is loosely attached to a door instance through a property set. Apart from explicit (if indirect) relationships that can be found in the IFC model, other, implicit relationships networks can be extracted from such models. A prominent example are room connectivity graphs which can be used to check building code compliance or minimize evacuation times of buildings. A requirement for query mechanisms to allow a query of an arbitrary room to the nearest exit is its ability to run recursively through the network of room nodes that are connected by edges (representing doors etc.). Enabling end-users to retrieve answers to common questions such as “How many windows are on this floor?”, “What is the fire rating of all external doors?” and “What is the shortest way to the exit?” without requiring intimate knowledge of a complex BIM schema such as the IFCs is the use case of the proposed language.

By providing a modular architecture, more functionality will be integrated in future. Through the addition of query extensions the base language will allow more complex uses. Prospective extensions include spatial reasoning operators or domain specific operators that e.g. require the intermediate computation of model properties by means of procedural functions or complete external simulations which are propagated back into the model.

4. Implementation

In our prototype implementation, the model is based on the Eclipse Modeling Framework (EMF). The model specifications are described in the XML Metadata Interchange (XMI). XMI is a standard for exchanging metadata via Extensible Markup Language (XML) and integrates the industry standards Extensible Markup Language (XML), Unified Modeling Language (UML) and Meta Object Facility (MOF). UML and MOF both are maintained by the Object Management Group (OMG).

The bimserver.org platform already provides some means to extract partial building information models from a repository. Selections can be made by entering an Object ID, a Global Unique ID, or all instances of a selected entity in the IFC schema. It is also possible to create custom queries by writing Java code, however the threshold to actually use this feature is high and the learning curve steep. Because of that and because the bimserver.org is an Open Source project we integrate a Domain Specific Language (DSL) that wraps the underlying querying mechanisms and hides the low-level technicalities from end-users into the bimserver.org platform.

ANTLR (Another Tool for language Recognition) is a tool used in the construction of language tools. It can be used to implement Domain Specific Languages. ANTLR reads a language description file called a grammar and generates source files and auxiliary files. Most uses generate a lexer and a parser. A lexer reads an input stream and divides it into tokens. The parser reads a token stream and matches phrases in a target language. Eclipse, the ANTLR plugin for the Eclipse SDK and an IFC to EMF converter are the tools we use to develop this new domain specific language. By using the IFC to EMF converter, integrating this new framework into the BimServer project will be a straightforward task, yet performance differences between in-memory and on-disk models cannot be estimated at this time of the development.

In the next paragraph the Backus-Naur Form notation to describe the syntax of the proposed Domain Specific query Language is provided. Note that the specification provided currently limited mainly to the ‘select’ part of the language features, whilst ‘create’, ‘update’ and ‘delete’ are still work in progress

```
         <bimql> ::= <create> | <select> | <update> | <delete>
        <create> ::= "CREATE" "?" <ident> ("SET" <leftterm> "=" <rightterm>)+
        <select> ::= "SELECT" "?" <ident> <statement>+
        <update> ::= "UPDATE" "?" <ident> <statement>+ ("SET" <leftterm> "="
                         <rightterm>)+
        <delete> ::= "DELETE" "?" <ident> <statement>+
     <statement> ::= <wherestatement>
<wherestatement> ::= "WHERE" <expression>
    <expression> ::= <realtion> ("AND" | "OR" <relation>)*
      <relation> ::= <leftterm> ("==" | "!=" | "<" | "<=" | ">=" | ">") <rightterm>
      <leftterm> ::= "?" <ident> (("." | ".*.") (<ident> | "ENTITYTYPE" |
                         "ATTRIBUTENAME"))+
     <rightterm> ::= ('"' <ident> '"') | <number>
         <ident> ::= <letter> (<letter> | <digit>)*
        <number> ::= <digit>+
        <letter> ::= ("a".."z") | ("A".."Z")
         <digit> ::= "0".."9"
```

Figure 1: Prelimenary BNF of the proposed query language

The first rule is the ‘bimql’-rule, denoting the start of the query. This rule enables the user to choose the action to be carried out. The top-level choice are the four actions ‘create’, ‘select’, ‘update’ and ‘delete’. Only one action can be selected at a given time. In all four cases the token is followed by a variable designated by the question mark sign (this syntax is inspired by other languages such as SPARQL). Variable names are freely chosen by the user and will be later assigned with lists of query results that are returned to the end-user. The ‘create’-rule is followed by the SET-token. This token, the ‘leftterm’-rule, the assignment-token and the ‘rightterm’-rule make it possible to assign properties to the entity created. More than one property can be assigned in a single query execution. The variable in the ‘select’ rule is followed by one or more statements. Statements make it possible to specify what to select and to narrow the selection. The ‘update’ acts as a hybrid rule in which a first statement selects the entity or graph node to be selected followed by a SET-token to determine its new value. Currently the only statement specified is a ‘where’ statement. Every ‘where’ statement starts with a token that identifies it and is followed by an expression-rule. An expression is a single relation or a combination of several relations. If more relations are specified within one expression these relations combined using the ‘OR’-token or the ‘AND’-token. These tokens indicate a dis- or conjunction between the relations. The relation-rule is specified by a ‘leftterm’- and a ‘rightterm’-rule and a collection of operator-tokens that separate them. The ‘leftterm’-rule points to the property or attribute to be changed or is involved in any other actions. The ‘leftterm’-rule starts with the variable defined earlier by the user. It is followed by the "."-token or the ".*."-token. Similar to EXPRESS language rules, the "."-token indicates a direct relation between its operands. In addition to these common combinations typically an entity name followed by an attribute (e.g. ‘IfcDoor.GlobalId’), the ".*."-token indicates an indirect relation between its operands. The ‘*’ asterisk is a placeholder that allows the matching of relation chains of arbitrary depths. This allows shortcuts and abbreviations in queries to act on nodes that are connected only indirectly via several edges and nodes in between them. A frequent example of such chains are properties in property sets assigned to entities (see figure 2). In future versions of the language, frequently used patterns will also be added as explicit operators (for example, the PSet case might be directly addressed with ‘[Entity].hasProperty’ constructs). Such explicit domain specific query operators would not only be syntactic sugar, but would also limit the search and graph-traversal and –matching scope, which is expected to yield performance enhancements esp. on larger models. These to operators can be followed by an identifier, the ‘ENTITYTYPE’- or ‘ATTRIBUTENAME’-tokens. The identifier can be used to base actions on specific properties or attributes found in a building information model. The ‘ENTITYTYPE’- and ‘ATTRIBUTENAME’-tokens can be used to specify the type of an entity (for example IfcDoor) or the name of an attribute (for example ‘OwnerHistory’ or ‘OverallHeight’). The ATTRIBUTENAME-token makes it for example possible to return all entities which have a ‘height’-attribute. This allows for both schema-level and instance-level query operations and is future proof for later versions of the IFC model. The ‘rightterm’-rule for the assignment of comparison can be a string. Numeric strings will be automatically matched to number types such as double or integer through conversions.

5. Examples

The development of this new Domain Specific Language is still in progress. In this section we will present a few examples in which this new language is used. These examples will give an insight in the basic structure of the framework.

The first examples show how to select only those building elements that satisfy certain criteria. We will for example select only those spaces that have a floor area larger than 20 square meters. This example also serves as an illustration as to why a domain specific language that provides syntactic simplifications compared with a general purpose language is useful for complex models such as the IFC. The relation of an entity (IfcSpace in this example) with its properties that go beyond the few direct attributes defined in the core schema spans constitutes a complex sub graph. This requires several graph network 'hops' or nested iterations in procedural programming approaches and nested joins in traditional SQL based query languages.

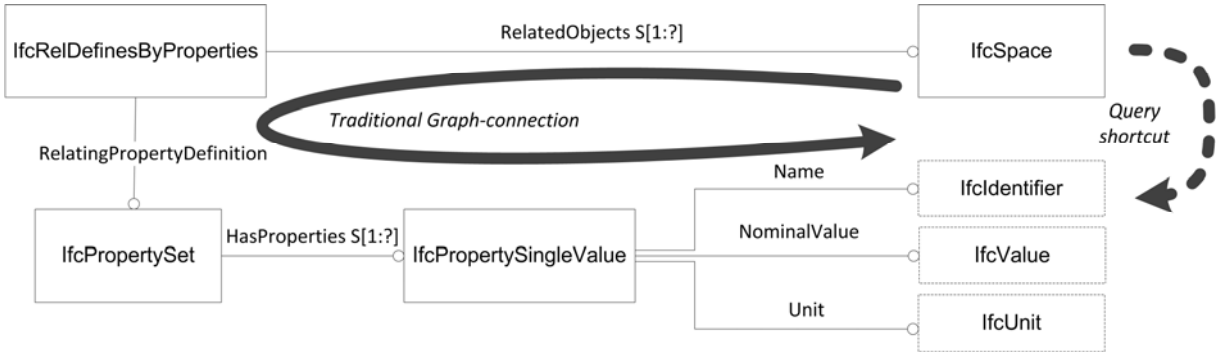

Figure 2: Query Shortcut against the Traditional Graph-connection

Although the information in an IFC model is organized very well, some steps are needed to retrieve certain information. This new framework streamlines that process.

5.1 Read

The first examples will retrieve parts of the IFC model. Parts of an IFC model can be all the windows, the first floor or all the columns, but a part can also be a list of numbers, for example all the doors and their dimensions. The first example returns all the spaces.

```
SELECT ?mySpace WHERE
    ?mySpace.ENTITYTYPE == "IfcSpace"
```

The second example returns all the spaces which area is larger than 20 square meters. Notice the .*. operator. It indicates that the operand it follows has a relation with the operand it precedes. In the case provided, it will match instances of the 'IfcSpace' entity which have a 'NetFloorArea' property assigned to them by an IfcElementQuantity property set. Matching such a sub graph might involves the use of (implicit) inverse relationships or the traversal of the explicitly present RelatedObjects/RelatingPropertyDefinition objectified relations in an inversed edge direction.

```
SELECT ?mySpace WHERE
    ?mySpace.ENTITYTYPE == "IfcSpace"
    ?mySpace.*.NetFloorArea > "20"
```

The last example returns the floor area of a space which GlobalID is known. Notice the statement in the second line. This statement is true when the value stored in the 'Name' attribute equals 'NetFloorArea'. This statement is also true when the name of an attribute equals 'NetFloorArea'.

```
SELECT ?myNetFloorArea WHERE
    ?myNetFloorArea.NAME == "NetFloorArea" FROM
        ?mySpace WHERE
```

```
    ?mySpace.GlobalID == "3Dn6BYWjfErxE1JocogMGQ"
```

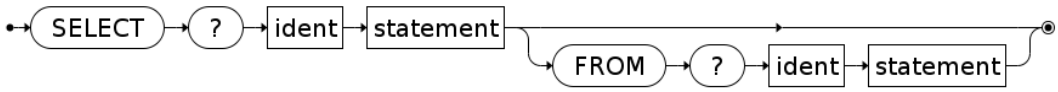

Figure 3: Select Syntax Diagram

5.2 Delete

It will also be possible to delete elements from an IFC model. The first example deletes one door. The GlobalID of that door is known.

```
DELETE ?myDoor WHERE
    ?myDoor.GlobalID == "3Dn6BYWjfErxE1JocogMGQ"
```

The second example deletes all doors whose height is more than 2 meters.

```
DELETE ?myDoor WHERE
    ?myDoor.ENTITYTYPE == "IfcDoor"
    ?myDoor.OverallHeight > "2"
```

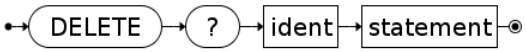

Figure 4: Delete Syntax Diagram

5.3 Update

The update feature will make it possible to change the values of an attribute or property. In the next example the name of a space is altered.

```
UPDATE ?mySpace WHERE
    ?mySpace.GlobalID == "3Dn6BYWjfErxE1JocogMGQ"
        SET ?mySpace.NAME = "Kitchen"
```

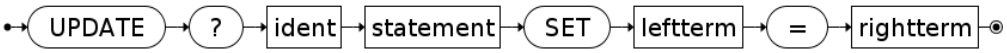

Figure 5: Update Syntax Diagram

6. Summary and Outlook

In this paper we introduced the on-going developments of a domain specific query language for the selection, addition and update of partial aspects in building information models. We have introduced our conceptual approaches and have outlined some of the requirements we have identified. We have also shown some examples in which our new framework is applied. Early stages of a prototypical implementation of the proposed language look promising and the general interest and positive feedback to the work so far will drive the future developments. The language will be designed and implemented on top of the bimserver.org platform even though it will be generic enough to be adapted in other implementations of IFC-based modeling and development tools. At this stage, it is too early to provide indicative

performance measures. The ongoing development is focused on the identification of frequent use patterns in order to create syntactic shortcuts (see the 'hasProperty' example in section 4), the efficient implementation on the bimserver platform and experiments with fuzzy natural language enhancements that would potentially increase usability by allowing end-users to operate with non-technical vocabulary ('walls', 'wall', 'Wand', 'muur' covering IfcWall and IfcWallStandardcase). We are investigating the inclusion of user feedback from the research and practice communities and will aim at a formal specification proposal in the near future.

References

Adachi, Y. (2003). Overview of Partial Model Query Language. In proceedings of the 10th ISPE International Conference on Concurrent Engineering (ISPE CE 2003), 549-555.

Prudhommeaux, E., Seaborne, A. (2008). SPARQL query Language for RDF. Available at: http://www.w3.org/TR/rdf-sparql-query/ (Accessed December 7, 2011).

Beetz, J., van Berlo, L.A.H.M., de Laat, R. and Bonsma, P. (2011). Advances in the development and application of an open source model server for building information. In proceedings of the 28th International Conference of CIB W78.

Borrmann, A., Beetz, J. (2010). Towards spatial reasoning on building information models. In proceedings of the 8th European Conference on Product and Process Modeling (ECPPM), 1-6.

Borrmann, A., Rank, E. (2009). Topological analysis of 3D building models using a spatial query language. Advanced Engineering Informatics, 23(4), 370-385.

Borrmann, A., van Treeck, C., Rank, E. (2006). Towards a 3D spatial query language for building information models. In Proceedings of the 11th Int. Conf. on Computing in Civil and Building Engineering (ICCCBE-XI).

Eastman, C., Lee, J., Jeong, Y., Lee, J. (2009). Automatic rule-based checking of building designs. Automation in Construction, 18(8), 1011-1033.

Eastman,C. (1999). Building product podels: Computer environments supporting design and construction CRC Press.

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

Hussmann, H., Zschaler, S. (2004). The Object Constraint Language for UML 2.0 - Overview and assessment. Upgrade Journal, 5(2).

Kriegel, A., Trukhnov, B. (2003). SQL bible. Wiley Publishing, Inc.

Lee, G., Eastman, C., Sacks, R. (2003). GT PPM user manual. Available at: http://dcom.arch.gatech.edu/gtppm/dn/GT%20PPM%20USER%20MANUAL_r4_1.pdf (Accessed December 24, 2011).

Lee, G., Sacks, R., Eastman, C. (2007). Product data modeling using GTPPM — A case study. Automation in Construction, 16(3), 392-407.

Martin, J. (1983). Managing the database environment. Prentice Hall.

Parr, T. (2007). The Definitive ANTLR Reference. Pragmatic Bookshelf.

Rattz, j., Hayes, D. (2009). Pro LINQ language integrated query in VB 2008. Apress.

Warmer, J., Kleppe, A. (2003). The Object Constraint Language. Addison Wesley.

Weise, M., Katranuschkov, P., Scherer, R. (2003) Generalised Model Subset Definition schema. In Construction IT: Bridging the Distance, Proceedings of the CIB-W78 Workshop