BimQL

an open domain specific query language for building information models

Mazairac, L.A.J.

Award date:
2012

BimQL

An Open Domain Specific Query Language for Building Information Models

Ludovicus Antonius Josephus (Wiet) Mazairac
November 2012

Project website:
http://bimql.org

Eindhoven University of Technology
Faculty of Building and Architecture
Design Systems group

Graduation committee:
Prof. dr. ir. Bauke de Vries (chairman)
Dr. Dipl.-Ing. Jakob Beetz
Ir. Joran Jessurun

1 Abstract

This report presents the design, development and evaluation of BimQL, an open, domain specific query language for Building Information Models. The query language is intended for selecting and updating data stored in Industry Foundation Class (IFC) models. Even though some partial solutions already have been suggested, none of them are open source, domain specific, platform-independent and implemented at the same time. This report presents an overview of existing approaches, the design decisions, the implementation on top of the BimServer.org platform, which is an open source model server, functionality and tests. The execution of example test cases show the general feasibility and scalability of the approach chosen.

2 Table of Contents

3 Introduction

During the design and construction process of a building many actors are involved. Some and probably the best known actors are the architect, the construction engineer and the contractor. Those parties generate very much information while working on a new design and the degree of success of the design and construction process depends strongly on the efficient exchange of that information between the different parties.

Mankind has been building for thousands of years and until recently the only way to exchange information was by sending each other drawings. Although many complex buildings have been completed successfully this way, some drawbacks exist. The availability of the drawings is a problem. Not all drawings will be available to all parties at any point of time during the building and construction process. The draftsman most likely stores his drawings in a file cabinet at the office. When requested, a copy of a drawing will be sent, however the office must be open and someone has to be available to send the drawings. The second problem might be the time it takes to send a drawing. If the drawings are sent by post, it takes some time before they arrive and therefore they might already be outdated on arrival. Updating all the drawings when the design has been altered is another problem. It takes much time and effort to change all drawings and to check whether the altered parts of the design are not conflicting with the unchanged, already existing parts.

With the introduction of digital drawings and electronic mail some of these problems were solved. Although the digital drawings still need to be sent, they arrive instantly and are therefore not outdated at the time of reception. However it still takes much time and effort to change all the individual digital drawings and without contacting other parties it is not certain the currently owned digital drawing is the most recent one.

These problems were solved with the introduction of model servers. A model server is a computer on which all information needed during the design a building process is stored. This computer can be accessed at any time and from anywhere through the internet. If an actor needs to change the design, first the most recent model needs to be downloaded, then it can be altered and when the changes have been

made it can be uploaded again to the model server. The actor can use any computer program that is able to import and export a building information model. Many computer programs used in the building industry today can import a model from and export a model to the Industry Foundation Classes (IFC) data model standard. This standard is an open standard and, developed to describe building and construction data and therefore frequently used in the process of building information modeling. Any computer is controlled by software. The software developed by the BimServer.org project turns any computer, server or standalone, into a Building Information Modelserver. The BimServer.org software does not store files, it stores models, which are created or changed when an actor uploads a new file. Additions and alterations are compared with the model already present on the server and a warning can be sent when there's a conflict between the old and the new design. The BimServer.org software, which is free to use and open source, can be used in conjunction with the IFC standard.

4 Problem Statement

While a large amount of information is generated by different parties during the design and building process, each actor will only be interested in a small part of that information at a certain point in time.

The architect for example might be interested in the overall design (Figure 4-1), the contractor needs to know how many doors of a certain type should be ordered and the building service engineer is mainly interested in the heating and ventilation system (Figure 4-2).

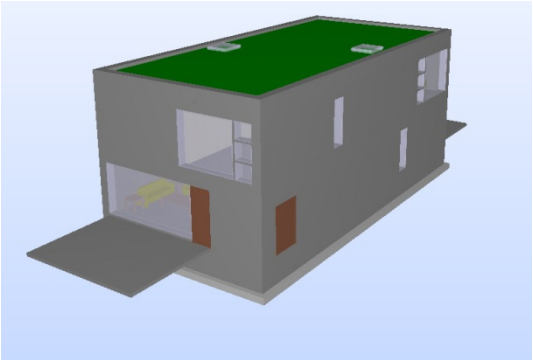


Figure 4-1: Overall design.

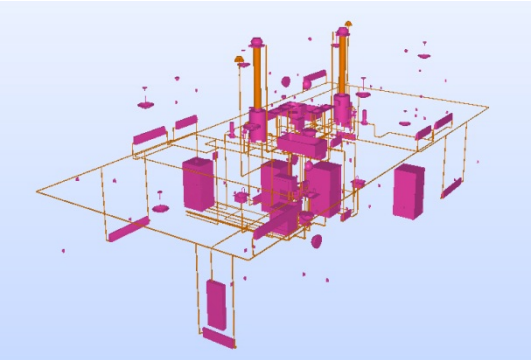


Figure 4-2: Building systems.

4.1 Paper Drawings

Simultaneously to the introduction of new drawing- and modeling-techniques, new methods for retrieving specific parts from a collection of data were introduced. In the period when drawings were the main method for storing and exchanging information, file cabinets and indexes were used to store and retrieve information. Those means were not very flexible. While it was possible to retrieve a drawing, how to search for it was dictated by the method the file cabinet and the index was organized. It was also impossible to combine information from multiple drawings instantly. First the required drawings were collected and then a draftsman could produce a new drawing which combined the information from those drawings. Compared to the means available today, this process took a long time.

4.2 Digital Drawings

With the introduction of digital drawings that process improved. Digital drawings were essentially stored the same way paper drawings were stored, however instead of a file cabinet a directory structure on a digital data storage device was used. Retrieving a particular drawing was much more convenient. Multiple drawings could be checked for a certain keyword instantly. Keywords could for example clarify by whom the drawing was created or which part of the building was described and provide information about different revisions. Information of multiple drawings could be combined into one drawing by copying parts from existing drawings and pasting them into the new drawing. Although this method is much quicker than the methods available before, the absence of any form of automation still makes this method labor-intensive and error-prone.

4.3 Building Information Model

With introduction of the model server the problem of data being scattered disappeared. Instead of searching in multiple digital or non-digital drawings, all data was stored in one single model, the Building Information Model. Still most actors are only interested in a small part of the data stored on a model server. This problem is not solved by only centralizing the storage of all data. As the use of a file cabinet was simplified by the corresponding index, the use of the Building Information Model could be facilitated by a method, which would provide the possibility to retrieve part of the model.

4.4 Fundamental Concepts

Various approaches have been proposed for selecting and filtering data from a server on which a building information model is stored. No solution however implements all fundamental concepts we identified at the start of this project. We believe any solution should implement those and therefore any solution should be;

- versatile by the possibility to define custom queries
- intuitive and user-friendly
- platform independent
- open source
- implemented

4.4.1 Definition of Custom Queries

When an approach does not allow the definition of custom queries the usefulness of that approach is limited. While a fixed subset may provide a solution when faced with common design situations, these fixed subsets are potentially useless when confronted with unanticipated design questions.

4.4.2 Usability

An approach can offer the required functionality, however it is still not useful when that functionality cannot be accessed in a pleasant and straightforward way.

4.4.3 Platform Independency

The building industry uses many software packages and applications during the design and construction process. A solution, interwoven with and depending on a specific application, can only be used from within that specific application. Platform independency eliminates that problem and increases user-friendliness at the same time, because the end-user is given one tool only to get familiar with.

4.4.4 Open Source Philosophy

Although the ability to define custom queries should do in most situations, a software development team cannot anticipate on all design and construction challenges. Therefore others, developers or end-users, should be able to adapt or add new features to an already existing solution. This is not possible when depending on proprietary software.

4.4.5 Implementation

Naturally the concepts described above can only be enjoyed after they are implemented. This might sound self-evident, however this report is probably only useful for other system engineers, while the solution developed at the end of this project will actually be useful for designers and other end-users.

5 Requirements

Next to the fundamental concepts described in the previous chapter more specific requirements need to be formulated in order to set the scope for this tool.

5.1 Create, Read, Update and Delete

It is very likely that, while managing data, objects not only need to be read, but also need to be created, updated or deleted. It might for example be necessary to change the manufacturer of the windows or maybe the color of the walls on the third floor need to be changed by the architect. Therefore the solution to be developed should adhere to the CRUD-principle. The letters which form the acronym CRUD stand for create, read, update and delete, which are the four most fundamental functions used while managing data.

5.2 Relations

An IFC (Industry Foundation Classes) model is an EXPRESS based entity-relationship model while EXPRESS is a standard data modeling language for product data. So basically, an IFC model is a collection of objects related to each other. The method under development should enable the end-user to retrieve specific objects based on its relation to other objects. It should for example no longer be difficult to retrieve all windows related to a specific wall.

5.2.1 Attributes and Properties

Sometimes the relations between IFC entities can be traced back to connections found within a real building. For example the relation between a window and wall in an IFC model is quite similar to the connection found in the real world. The window fills an opening and the opening is part of a wall (Figure 5-1, 5-2, 5-3 and 5-4).

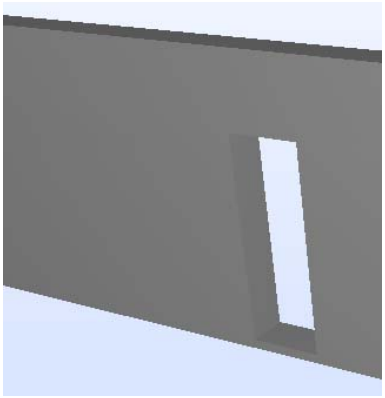


Figure 5-1: Wall.

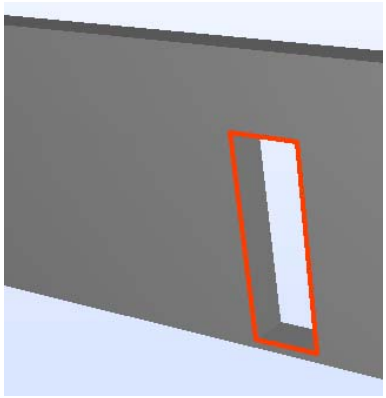


Figure 5-2: Opening.

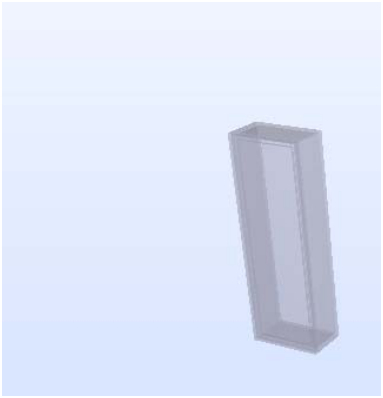


Figure 5-3: Window.

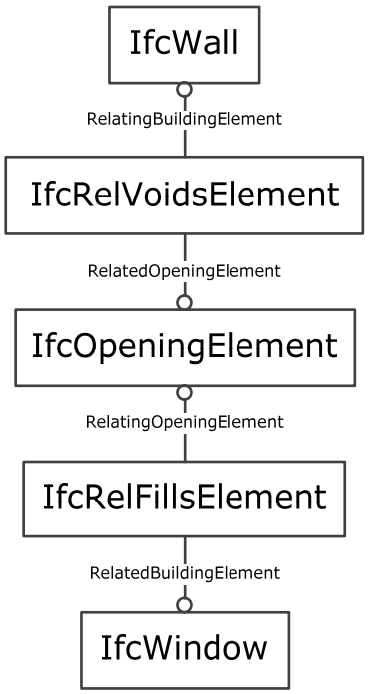


Figure 5-4: Connection between IfcWall and IfcWindow.

Other relations are more difficult to trace back. Although the dependency between an object and one of its properties might seem straightforward, this dependency is much more complicated than the relation between an object and one of its attributes (Listing 5-1).

```
1. ENTITY IfcWindow
2. SUPERTYPE OF (IfcWindowStandardCase)
3. SUBTYPE OF (IfcBuildingElement)
4.     OverallHeight
5.     OverallWidth
6. END_ENTITY
```

Listing 5-1: OverallHeight and OverallWidth as attributes of IfcWindow.

The IFC schema differentiates between attributes, directly attached to an object, and properties, grouped in a propertyset and assigned to the object by a number of relations (Figure 5-5). To find the color of a building element (a color is a property, not an attribute) not one, but multiple nodes present within the network need to be passed first. In this case the wall would be red when IfcIdentifier equals color and IfcValue equals red (Figure 5-5).

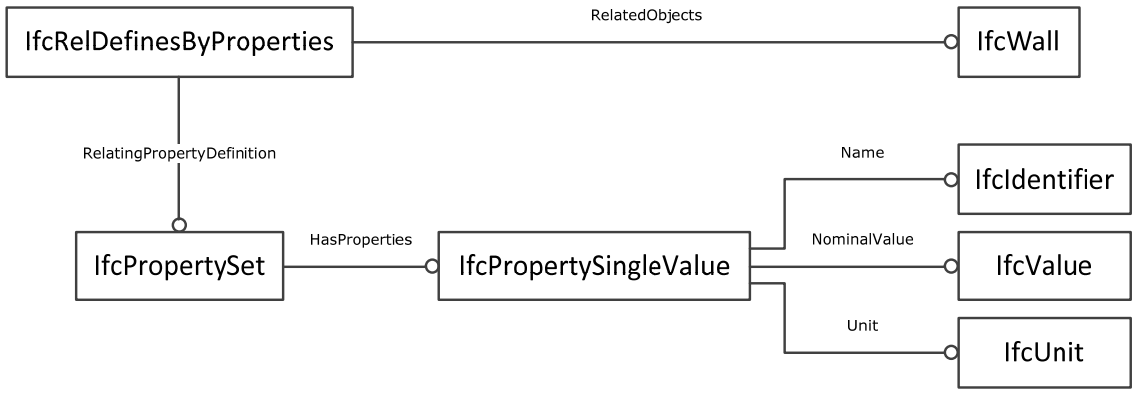


Figure 5-5: Connection between a wall and one of its properties.

A non-expert user without any knowledge of the IFC data model is not familiar with the differences between attributes and properties. Therefore, instead of the non-expert user, the method under development should make that distinction. When a color needs to be retrieved, the value of a property will be retrieved automatically, not the value of an attribute.

5.2.2 Recursive Queries

The steps required while retrieving an attribute or property are known in advance and therefore it is possible to automate this process. However sometimes it is not possible to know beforehand which or how many steps need to be taken to retrieve a relationship. This might happen when creating a room connectivity graph and searching for the shortest route outside (Figure 5-6).

Figure 5-6: Complex room connectivity graph.
All arrows are pointing towards the exit.

For such graph-like structures, recursive queries are necessary in order to find information, enabling end users to retrieve information even from objects connected to other objects through an arbitrary number of relationship edges.

6 Related Work

Over time a wide range of querying approaches has been developed that allows extracting and editing part of a data collection. Approximately two categories can be distinguished. The first category includes the generic querying approaches, while the second category was developed to streamline the process of building information modeling. An overview of the different querying approaches is provided next.

6.1 Generic Querying Approaches

6.1.1 SQL

The Structured Query Language (SQL) is used by many existing database applications. SQL was designed for managing data in relational database management systems (RDBMS). It provides the possibility to create, read, update and delete records and therefore conforms to the CRUD principle. Listing 6-1, 6-2 and 6-3 show a table which is queried by SQL and the results of that query.

```
1. FirstName   Age
2. John        34
3. Pete        51
4. Andrew      19
5. Jack        29
6. Jim         73
```

Listing 6-1: Table with name and ages.

```
1. SELECT Table.FirstName, Table.Age
2. FROM Table
3. Where Table.Age>40
```

Listing 6-2: SQL query which selects all persons over 40.

```
1. FirstName   Age
2. Pete        51
3. Jim         73
```

Listing 6-3: Result of the SQL query.

Although SQL is the most widely used query language, the possibilities for querying and modifying a building information model or IFC data model are limited. SQL was designed for managing data stored in tables and hence the data stored in a building information model, which is object based, needs to be mapped to a table based storage system before SQL can be applied to that data. The IFC data model consists of over 600 entities and thousands of related attributes.

Although nested and recursive queries can be formulated while using standard SQL, the complexity would increase rapidly beyond levels of feasibility, especially when the end user has no or little experience in programming.

6.1.2 LINQ

The Language Integrated Query (LINQ) is a Microsoft .Net Framework component that extends .NET languages with query expressions. LINQ can be used for querying various types of data, for example databases, XML documents, or even a list of processes being executed by the central processing unit within a computer. Listing 6-4 shows an example of a LINQ query and the result can be found in listing 6-5.

```
1.  Public Sub XLinq()
2.      Dim doc = XDocument.Load(dataPath + "nw_customers.xml")
3.      Dim result = doc.<Root>.<Customers>(0)
4.      Console.WriteLine(result)
5.  End Sub
```

Listing 6-4: LINQ query, which selects a customer.

```
1.  <CustomersCustomerID="ALFKI">
2.      <CompanyName>AlfredsFutterkiste</CompanyName>
3.      <ContactName>MariaAnders</ContactName>
4.      <ContactTitle>SalesRepresentative</ContactTitle>
5.      <Phone>030-0074321</Phone>
6.      <Fax>030-0076545</Fax>
7.      <FullAddress>
8.          <Address>ObereStr.57</Address>
9.          <City>Berlin</City>
10.         <PostalCode>12209</PostalCode>
11.         <Country>Germany</Country>
12.     </FullAddress>
13. </Customers>
```

Listing 6-5: Result of the LINQ query.

Currently LINQ can only be used in conjunction with .NET languages and although similar approaches for other languages have been proposed, a common standard will probably never become platform independent. The usefulness of LINQ, while querying a building information model, is limited by its generic nature. Because of its generic nature, intimate knowledge of the IFC data model is required and queries need to be written in an extensive programming language, not intended for an average end-user.

6.1.3 RDF

The Resource Description Framework (RDF) is a data format for representing information on the World Wide Web. RDF is often used to represent personal information, as well as to provide a means of integration for disparate sources of information. The SPARQL Protocol

and RDF Query Language (SPARQL) [16] is the most established RDF query language and has been officially standardized. A SPARQL query can be sent to multiple SPARQL endpoints, which are services that accept SPARQL queries and return results. Listing 6-6, 6-7 and 6-8 show data which is queried by SPARQL and the results of that query.

```
1. ab:richard ab:homeTel "(229) 276-5135" .
2. ab:richard ab:email "richard49@hotmail.com" .
3. ab:cindy ab:homeTel "(245) 646-5488" .
4. ab:cindy ab:email "cindym@gmail.com" .
5. ab:craig ab:homeTel "(194) 966-1505" .
6. ab:craig ab:email "craigellis@yahoo.com" .
7. ab:craig ab:email "c.ellis@usairwaysgroup.com" .
```

Listing 6-6: Names, phone numbers and email addresses.

```
1. SELECT ?craigEmail
2. WHERE
3. { ab:craig ab:email ?craigEmail . }
```

Listing 6-7: SPARQL query which selects specific email addresses.

```
1. craigEmail
2. "c.ellis@usairwaysgroup.com"
3. "craigellis@yahoo.com"
```

Listing 6-8: Result of the SPARQL query.

The ability to query multiple data repositories instantly allows linking data from different sources effectively. In a BIM context, such a mechanism would allow the ad hoc connection of building information models stored at different locations. For the design of BimQL, a number of features have been inspirational, including the syntactical means to specify and reference variables by the '?' token.

6.1.4 OCL

The Object Constraint Language (OCL) [9] is a language for precise textual descriptions of constraints which apply to graphical models captured in the Unified Modeling Language (UML).

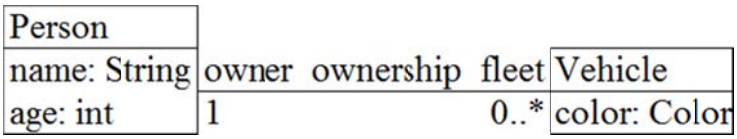


Figure 6-1: Relation between vehicle and owner.

```
1. context Vehicle
2. inv: self.owner.age>=18
```

Listing 6-9: The owner of a vehicle should be 18 or over.

6.2 BIM Querying Approaches

6.2.1 EQL

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) [20] proposed by Huang. EQL is a query language that is used to perform ad hoc queries on data stored in the STEP Part 21 file format. The STEP Part 21 file format, also called the STEP-File method, is the most widely used method for exchanging an information model based on EXPRESS, which is a standard data modeling language for product data. Unlike the STEP-File method, EQL is not widely used. The reason for this might be that EQL is not a closed language [10], which means the result of a query cannot be queried again. It would for example be impossible to first select all doors of a specific height and subsequently select the walls in which those doors are located.

6.2.2 PMQL

The Partial Model Query Language (PMQL) [1] proposed by Adachi aims to provide a general means for selecting, updating, and deleting partial model data. Among the clear benefits the language design has over generic query languages described in section 6.1, is the fact that it has been designed with IFC based models in mind and therefor the queries are fairly simple and straightforward (Listing 6-10).

```
1.  <select type="entity" match="IfcDoor" action="get">
2.      <cascades>
3.          <select type="attribute" match="OverallHeight" action="get"/>
4.      </cascades>
5.  </select>
```

Listing 6-10: The height of every doors is queried.

However, it currently does not provide the possibility to create or add model data to an existing building information model and its XML syntax would require additional tools to enable non-programmers to construct practical queries. For the design and specification of BimQL, PMQL has been an important influence. Among other things, this is reflected by the introduction of the ‘cascade’ rule, which has been proposed by Adachi to cope with recursive characteristics of IFC model graphs.

6.2.3 GTPPM

The Georgia Tech Process to Product Modeling (GTPPM) [13] 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 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 Information Delivery Manual (IDM) implementation details, it cannot automate the generation of an entire IDM. IDM describes the processes and the data required by identifying the steps of a building process, the information required by each step and the results of each step. It specifies where the process fits, the actors involved, the information created and consumed and the required software. Benefits of using GTPPM as a method to create an IFC IDM view include traceability and reusability.

6.2.4 GMSD

The Generalized Model Subset Definition (GMSD) [19] schema devised by Weise et al 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 designed to the support 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 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.

6.2.5 Solibri

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.

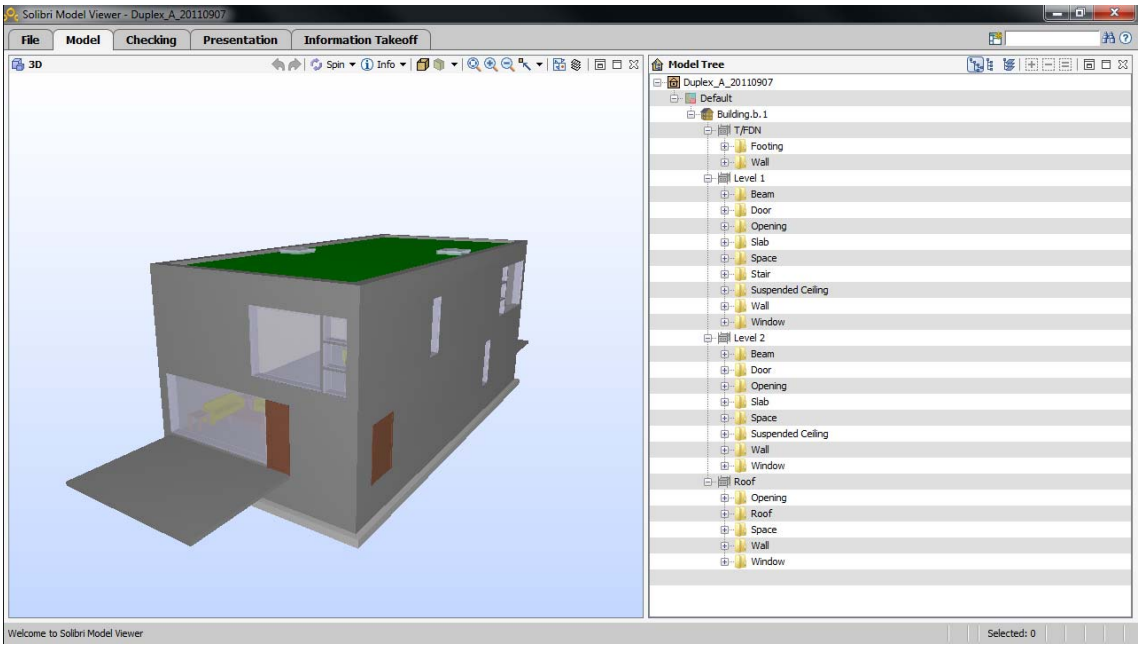


Figure 6-2: Solibri Model Viewer.

6.2.6 BimServer.org

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 the name of a class from the IFC schema (Figure 6-3). 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. For this reason and because the BimServer.org project is an open source project, we made the choice to integrate our Domain Specific Language (DSL) into the BimServer.org project. BimQL wraps the underlying querying mechanisms and hides the low-level technicalities from end-users.

Figure 6-3: BimSer.org simple query interface.

6.3 Conclusion

No suitable solutions have been found among those investigated. The generic querying approaches might be versatile enough to create the desired custom queries. However the downside of that versatility, and thus the lack of predefined IFC functionality, for example the possibility to select a property or attribute, is the complexity of those queries.

A querying approach which is based on IFC data models could be the solution, however only when all other requirements are met. Unfortunately no such solution has been found. PMQL for example has not been implemented, Solibri is not open source and the advanced query method from the BimServer.org platform is too complicated for most users.

7 Implementation Method

The Eclipse Integrated Development Environment (IDE), an Industry Foundation Classes (IFC) to Eclipse Modeling Framework (EMF) converter and the Another Tool for language Recognition (ANTLR) plugin for the Eclipse IDE were the main tools used during the development of BimQL. The EMF model is the core of the BimServer.org software and by using the IFC to EMF converter, which provides an independent and isolated EMF model, quick development was possible.

7.1 Model Driven Architecture

Model Driven Architecture (MDA) is a method for developing software. Instead of developing the source code itself, the programmer develops a model, which is used for automatically generating the source code. Although it can take much time to develop a system that automatically generates source code from a model, this method increases portability, productivity and cross-platform interoperability. The BimServer.org software is based on this approach. First the EXPRESS schema is converted to an EMF model. This model is then used to generate Java classes for communicating with the BimServer.org database. Some Java classes part of the BimQL project were also automatically generated. The BimServer.org Java classes which are based on the EMF Ecore model were used for generating some of the BimQL Java classes.

7.1.1 EXPRESS Schema

EXPRESS is a standard data modeling language for product data. While a building is a product, EXPRESS can be used to describe building information. A schema is a data model in a formal notation. The IFC specification consists of such a schema and describes a set of data types and their possible relationships (Listing 7-1).

```
1. ENTITY IfcDoor
2.     SUPERTYPE OF (IfcDoorStandardCase)
3.     SUBTYPE OF (IfcBuildingElement);
4.     OverallHeight: OPTIONAL IfcPositiveLengthMeasure;
5.     OverallWidth : OPTIONAL IfcPositiveLengthMeasure;
6. END_ENTITY;
```

Listing 7-1: IfcDoor EXPRESS schema.

7.1.2 EMF Model

The Eclipse Modeling Framework (EMF) can be used to develop a domain model. EMF is based on two meta-models; the Ecore model and the Gen model. The Ecore model (Listing 7-2) contains information about classes which are related to the data types and possible relationships, both described by the EXPRESS schema. The Gen model contains additional information for generating code, in this case the BimServer.org Java classes.

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 XML, Unified Modeling Language (UML) and Meta Object Facility (MOF). UML and MOF both are maintained by the Object Management Group (OMG).

```
1. <eClassifiers xsi:type="ecore:EClass" name="IfcDoor"
   eSuperTypes="#//IfcBuildingElement">
2.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallHeight"/>
3.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallHeightAsString"/>
4.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallWidth"/>
5.     <eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallWidthAsString"/>
6. </eClassifiers>
```

Listing 7-2: IfcDoor EMF model.

7.1.3 BimServer.org Java Classes

The BimServer.org Java classes are used to, among other things, store a BIM model to the database and manipulate objects already stored in the database. Each class contains ‘getters’ and ‘setters’, which are methods used to manipulate variables. The BimQL Java classes are based on those methods (Listing 7-3).

```
1. public class IfcDoorImpl extends IfcBuildingElementImpl implements IfcDoor {
2.     public double getOverallHeight() {
3.         return (Double) eGet(Ifc2x3Package.Literals.IFC_DOOR__OVERALL_HEIGHT,
   true);
4.     }
5. }
```

Listing 7-3: IfcDoor BimServer Java classes.

7.1.4 BimQL Java Classes

Although the BimQL Java classes are not generated from a model directly and hence they are not really a part of the Model Driven Architecture approach, indirectly these classes are still based on the EXPRESS schema and therefore worth mentioning here. These classes (Listing 7-4) establish a link between the developed query language and the BimServer.org Java classes, talking to the database in which a building information model is stored.

```
1.  public class SetAttributeSubIfcDoor {
2.      public void setAttribute() {
3.          ...
4.          else if (attributeName.equals("OverallHeight")) {
5.              ((IfcDoor) object).setOverallHeight(Double.parseDouble(attributeNewValue));
6.          }
7.      }
8.  }
```

Listing 7-4: IfcDoor BimQL Java classes.

7.2 ANother Tool for Language Recognition

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 (DSL). ANTLR reads a language description file called a grammar and generates source files and auxiliary files. Most users 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.

It is possible to write the lexer and the parser manually in probably any programming language, for example JAVA, however that would be time-consuming and error-prone. It is much more convenient to define the language structure only (Figure 7-5) and let ANTLR generate a lexer (Figure 7-6) and parser which are based on that structure.

```
grammar Verslag;

options {
    language = Java;
}

@header {
    package nl.wietmazairac.bimql;
}

@lexer::header {
    package nl.wietmazairac.bimql;
}

start: 'Select' STRING;
STRING: '"' ('a'..'z')+ '"';
WS: (' ') + {$channel = HIDDEN;};
```

Listing 7-5: Complete ANTLR grammar.

```
                default :
                    if ( cnt1 >= 1 ) break loop1;
                        EarlyExitException eee =
                            new EarlyExitException(1, input);
                        throw eee;
                }
                cnt1++;
            } while (true);

            match('\"');

            }

            state.type = _type;
            state.channel = _channel;
        }
        finally {
        }
    }
    // $ANTLR end "STRING"

    // $ANTLR start "WS"
    public final void mWS() throws RecognitionException {
        try {
            int _type = WS;
            int _channel = DEFAULT_TOKEN_CHANNEL;
            // D:\\Workspace\\BimQL\\src\\nl\\wietmazairac\\bimql\\Verslag.g:17:3: ( ( ' ' )+ )
            // D:\\Workspace\\BimQL\\src\\nl\\wietmazairac\\bimql\\Verslag.g:17:5: ( ' ' )+
            {
            // D:\\Workspace\\BimQL\\src\\nl\\wietmazairac\\bimql\\Verslag.g:17:5: ( ' ' )+
            int cnt2=0;
            loop2:
            do {
                int alt2=2;
                int LA2_0 = input.LA(1);

                if ( (LA2_0==' ') ) {
                    alt2=1;
                }


                switch (alt2) {
                case 1 :
                    // D:\\Workspace\\BimQL\\src\\nl\\wietmazairac\\bimql\\Verslag.g:17:6: ' '
                    {
                    match(' ');

                    }
                    break;
```

Listing 7-6: Part of the lexer in JAVA.

8 Examples

In this section some examples are presented in which BimQL is used. Besides the specification provided in the next section, these examples give an insight in the basic structure of the language.

8.1 Example 1

The first example (Listing 8-1) retrieves all rooted entities of the IFC model. Rooted entities, for example the IfcWall- and IfcDoor-entity, derive from the IfcRoot-entity and have an identity, while non-rooted entities, for example the IfcLine- and IfcVector-entity, do not have an identity and can only exist when related to a rooted entity.

```
1. Select ?Var1
```

Listing 8-1: Select all rooted entities.

8.2 Example 2

The second example (Listing 8-2) retrieves part of the IFC model. Parts of an IFC model can be all windows, the first floor or the columns, but a part can also be a list of numbers, for example the height of all the doors. In the second example the 'EntityType'-token is used to retrieve all doors present in the model.

```
1. Select ?Var1
2. Where ?Var1.EntityType = "IfcDoor"
```

Listing 8-2: Select all doors.

8.3 Example 3

The third example (Listing 8-3) returns all entities with an arbitrary name. You could also state that this example retrieves all objects which have a 'Name'-attribute.

```
1. Select ?Var1
2. Where ?Var1.Attribute.Name = "*"
```

Listing 8-3: Select all with an arbitrary name.

8.4 Example 4

The next example (Listing 8-4) retrieves a list of strings instead of a collection of objects. This example shows how to get the Global IDs of all the doors present in the model. The first two lines in the example select the doors. This was already demonstrated in example two. This list of doors is referred to by the '?Var1'-variable. The third line creates a new variable, '?Var2' and performs a query on '?Var1'.

A cascade connection, which connects the input and output of two systems, makes it possible to query a result which was generated by an earlier query. In this case the first two lines could and the last line could be seen as a two separate systems. The output or the result of the first two lines is the input for the last line. The query loops through all objects present in the '?Var1'-list. This '?Var2'-list contains the Global IDs of all doors.

Notice the absence of the 'Attribute'-token in the third line.

```
1. Select ?Var1
2. Where ?Var1.EntityType = ”IfcDoor”
3. Select ?Var2 := ?Var1.GlobalId
```

Listing 8-4: Select Global IDs of all doors.

8.5 Example 5

In the fifth example (Listing 8-5a) the 'Property'-token is used to retrieve a volume.

```
1. Select ?Var1
2. Where ?Var1.Attribute.GlobalId = "87d87dffn47a90z"
3. Select ?Var2 := ?Var1.Property.Volume
```

Listing 8-5a: Retrieve the volume.

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 IFC models. The relation of an entity with its properties that go beyond the few direct attributes defined in the core schema constitutes a complex sub graph. Instead of using the 'Property'-token, the volume of an entity could be retrieved by applying multiple 'cascade'-connections or 'cascade'-rules and navigate manually through the IFC model (Listing 8-5b).

```
1. Select ?Var1
2. Where ?Var1.Attribute.GlobalId = "87d87dffn47a90z"
3. Select ?Var2 := ?Var1.Attribute.IsDefinedBy
4. Select ?Var3 := ?Var2.Attribute.RelatingPropertyDefinition
5. Select ?Var4 := ?Var3.Attribute.HasProperties
6. Where ?Var1.Attribute.Name = "Volume"
7. Select ?Var5 := ?Var4.Attribute.NominalValue
```

Listing 8-5b: Retrieve the volume without the 'Property'-token.

Using the 'Property'-token as a shortcut (Figure 8-1) is much more convenient, requires less knowledge of the IFC data model specification and was one of the main reasons for developing BimQL.

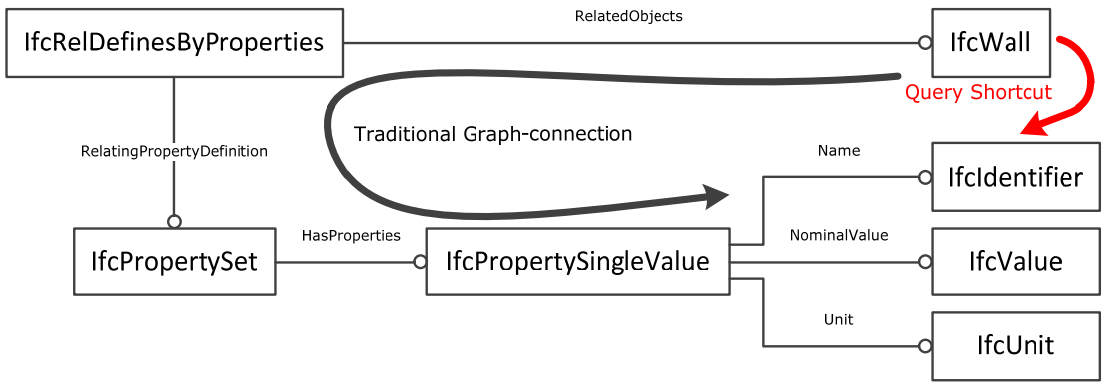


Figure 8-1: Query Shortcut against the Traditional Graph-connection.

8.6 Example 6

Example six (Listing 8-6) shows how to select all objects related to a specific object. Line 2 selects a specific object, based on a Global ID. The third line selects all related objects, which are three or less hops away. The '+'-sign indicates, that 'IfcRelDefines'-objects will also be selected.

```
1. Select ?Var1
2. Where ?Var1.Attribute.GlobalId = ”1eHJakbVPEdf9cGdMXVBAY”
3. Select ?Var2 := ?Var1.*(3+)
```

Listing 8-6: Select related objects.

8.7 Example 7

Example seven (Listing 8-7) selects all walls related to one space. First line 1 and 2 select one specific space (Figure 8-2). Line 3 selects all related elements and line 4 retrieves the walls from all the elements selected in line 3 (Figure 8-3).

```
1. Select ?Var1
2. Where ?Var1.Attribute.GlobalId = ”1eHJakbVPEdf9cGdMXVBAY”
3. Select ?Var2 := ?Var1.*(3)
4. Where ?Var2.EntityType = ”IfcWall”
```

Listing 8-7: Select walls related to one space.

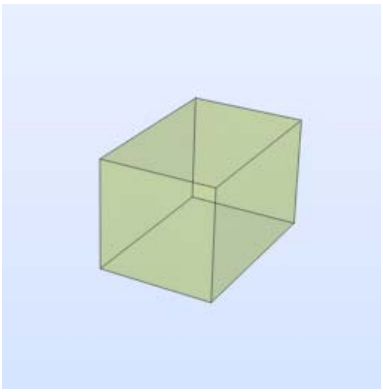


Figure 8-2

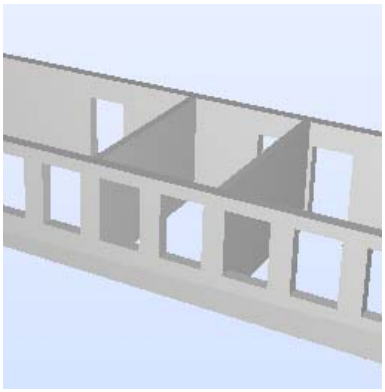


Figure 8-3

8.8 Example 8

The final example (Listing 8-8) shows how to change the value of an attribute. The first two lines select the object which attribute value needs to be changed. The third line actually changes the value.

```
1. Select ?Var1
2. Where ?Var1.Attribute.GlobalId = ”1eHJakbVPEdf9cGdMXVBAY”
3. Set ?Var1.Attribute.Description := ”RedDoor”
```

Listing 8-8: Change color of the door.

9 BimQL Specification

In this part the BimQL language specification will be presented. At the end of this part in listing 9-1 the Backus-Naur Form (BNF) is provided. While the first part of this chapter explains the specification in more detail, the BNF provides an overview of BimQL. The BNF describes the most recent version of the grammar, in which double quotes have been added to the 'DQSTRING'-token and in which the 'DOTSTRING'-token is introduced.

Note that the specification provided is currently limited to the 'select' and 'set' parts of the language features, while 'create' and 'delete' might be developed in the future. The implementation of the latter two language features require a significant amount of additional research and software development, because the consequences of these operations are far from straightforward. For example, even though the deletion of a single window could be achieved by issuing a simple query, the extraction would have to incorporate additional housekeeping and garbage collection to ensure model integrity. Such depending operations include the deletion of all its representations, its openings and its indirect dependencies such as window profiles and material specifications in order to avoid the presence of unreferenced entities while the model evolves.

Like any other grammar, The BimQL specification consists of a number of rules. By explaining the rules, it will become clear how BimQL operates and what it is capable of.

9.1 Bimql

The first rule is the 'bimql'-rule (Figure 9-1), denoting the start of the query. This rule enables the user to choose the 'select'-rule. You might expect other rules at this location, for example 'set'. However the syntax is designed in such way, you first have to select the items which need to be altered.

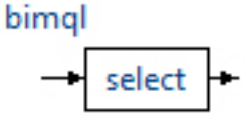


Figure 9-1: Bimql Syntax Diagram.

9.2 Select

Stating the ‘select’-rule (Figure 9-2) is called by the ‘bimql’-rule is not entirely correct, although this might sound obvious. The ‘bimql’-rule consists of only one nonterminal, the ‘select’-nonterminal, and two terminals, the start- and endpoint. The ‘select’-nonterminal is described by the ‘select’-rule or ‘select’-diagram. Instead of the ‘select’-rule begin called by the ‘bimql’-rule, the ‘select’-nonterminal is actually replaced by the ‘select’-rule or diagram.

First the ‘Select’-token is followed by a variable. Variable names can be chosen freely by the user and will be assigned with lists of query results. Those lists can be used again at various locations. They can for example be returned to the end-user or can be queried again in the ‘cascade’-rule. The variable can be followed by a ‘where’-rule, a ‘cascade’-rule or a ‘set’-rule. The first narrows the selection, the second selects related entities and the third can change the attributes of the selection.

Figure 9-2: Select Syntax Diagram.

9.3 Cascade

The ‘cascade’-rule (Figure 9-3) makes it possible to query a list of objects, created by a previous query. For example first the doors with a certain height are selected and next the ‘cascade’-rule selects the walls in which those doors are located. The ‘cascade’-rule assigns that list of objects to another variable, narrows the list, or selects some or all of the children related to the objects stored in it.

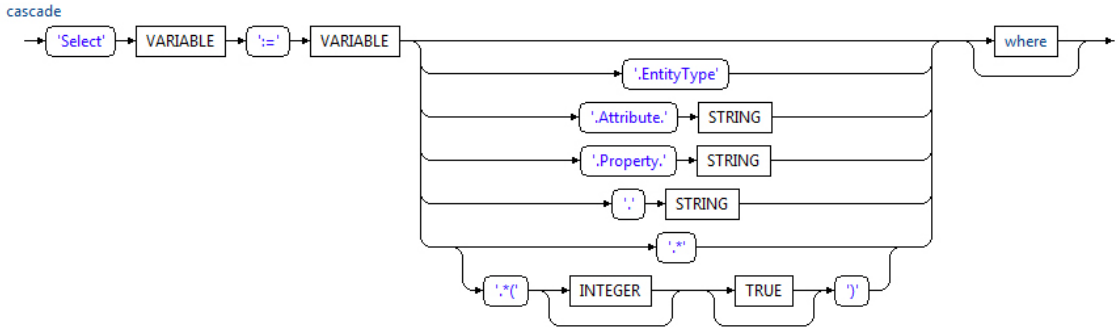


Figure 9-3: Cascade Syntax Diagram.

The first variable in this rule is used to store query results generated by this rule. The second variable contains the already existing list of objects. Consequently the existing list of objects is copied and assigned to the first variable. Next, the first option of six proceeds directly to the ‘where’-nonterminal. This nonterminal is optional and provides the possibility to narrow the list of objects just assigned to the second variable.

The next six options provide different methods for selecting children or properties of the objects contained by the second variable. First the ‘.EntityType’-option returns the IFC class of each queried object. The ‘Attribute.STRING’-option makes it possible to select a property of an object or to select another, related object. The ‘IfcDoor’-entity for example has a few attributes. One of those attributes is called ‘OverallHeight’ and another one is called ‘FillsVoids’. Actually the ‘IfcDoor’-entity is a subtype of the ‘IfcElement’-entity and inherits all the attributes owned by the ‘IfcElement’-entity. Therefore the ‘IfcDoor’-entity does not only own the ‘OverallHeight’- and the ‘OverallWidth’-attribute, however it also owns all attributes owned by the ‘IfcElement’-entity, for example the ‘FillsVoids’-attribute. Consequently to select the height of the door, the ‘STRING’-nonterminal is replaced with the string ‘OverallHeight’ and to select the voids which are filled by the door it is replaced with the string ‘FillsVoids’.

The ‘Property.STRING’-option is another possibility to retrieve more information about an object. It is important to realize that the IFC schema differentiates between attributes that are directly attached to an object as attributes, and properties, grouped in a propertyset and assigned to the object by a number of relations.

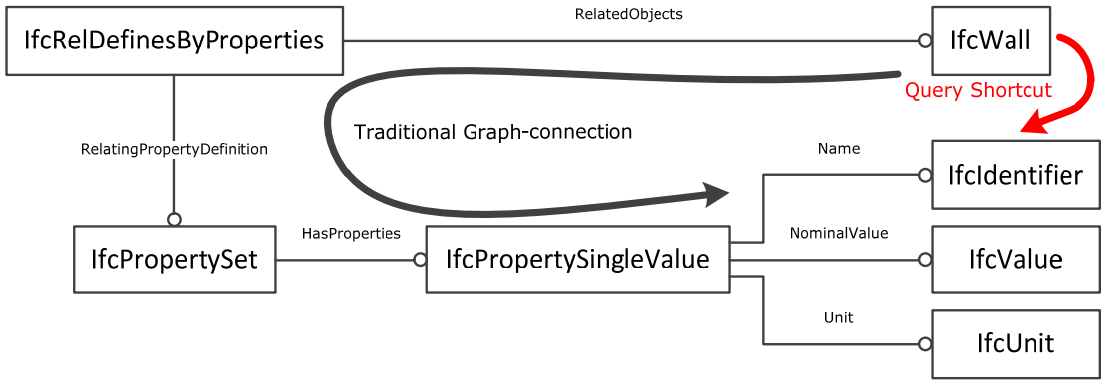


Figure 9-4: Query Shortcut.

It is possible to retrieve a property, part of a propertyset, by applying the ‘cascade’-rule multiple times. However this method is very cumbersome, time consuming and requires intimate knowledge of the IFC data model. The ‘Property.STRING’-option provides a convenient query shortcut (Figure 9-4), which reduces the required number of ‘cascade’-queries from four to only one.

The next option is the ‘.STRING’-option. This option combines the ‘Attribute.STRING’- and the ‘Property.STRING’-option. An end-user without knowledge of the IFC data model is unaware of the differences between attributes and properties. Therefore the end-user will probably not know that the height of a door is an attribute, while the color is a property. This option solves that problem. When this option is used, BimQL searches both the attributes and the properties for a match. This option has been implemented, however it is not yet completely error-free.

The ‘.*’-option selects all objects and properties, which are directly related to the objects stored in the second variable. As stated before, properties and objects are related to other objects by attributes. This option first checks which attributes are owned by an object and then it selects all objects and properties to which these attributes are pointing.

An alternative for the '.*'-option enables the end-user to select the depth of the 'cascade'-query and whether or not 'IfcRelDefines'-objects are selected. When the 'depth'-integer is set to zero, only the objects already stored in the second variable will be returned. When the 'depth'-integer is set to one, this option behaves like the '.*'-option, described in the previous paragraph and when the 'depth'-integer is set to two the children of the children stored in the second variable are also selected and so forth (Figure 9-5). The method for searching a graph described here is referred to as the breadth-first strategy.

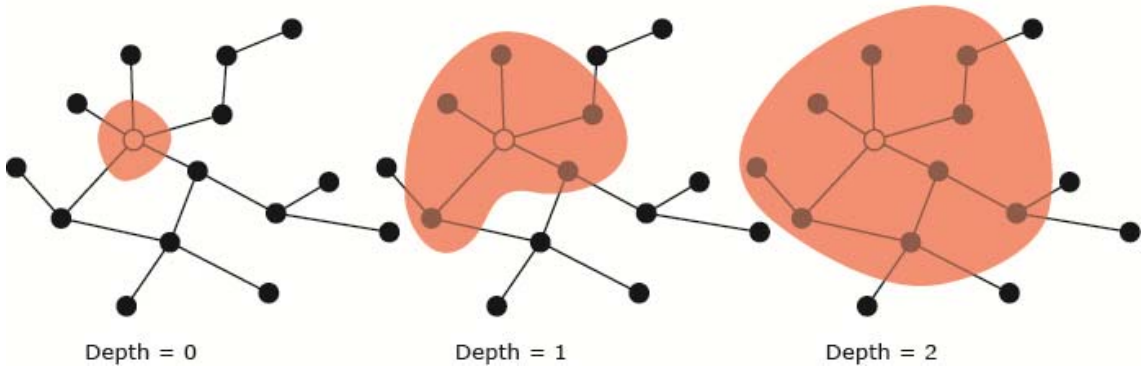


Figure 9-5: Depth Integer.

The end-user is provided with the option to ignore 'IfcRelDefines'-objects during a 'cascade'-query. The reason for this can best be explained by providing an example. If the end-user is looking for all objects related to a door, that user is probably looking for the wall in which it is located or the room to which it gives access. However that door might also have a color, which is a property, not an attribute. This property, the color can be returned by the 'cascade'-rule and other objects, related to that color, can also be returned by this rule. If 'IfcRelDefines'-objects are not ignored it is possible that all red objects are returned. So not only the wall and the room directly related to that door, also the red columns on another floor. Querying 'IfcRelDefines'-objects can be enabled by providing the optional 'true'-token, in this case a '+'-sign.

The Partial Model Querying Language (PMQL) designed by Y. Adachi was the main source of inspiration while developing the ‘cascade’-rule. The first approach was based on the ‘From’-clause defined in the SQL standard and although the resulting code was working quite well, I believe the ‘cascade’-approach is more intuitive, because while using the ‘from’-clause, the end-user needs to specify what to select first and then where to select it from. I believe this is inconvenient, especially when multiple ‘from’-rules are applied consecutively. The design of the ‘cascade’-rule solved this problem.

9.4 Where

The ‘where’-rule (Figure 9-6) provides both the ‘select’-rule and the ‘cascade’-rule with the possibility to narrow a selection. For each object, the statement is either true or false. When the statement is true, the object is kept and when false, the object will be removed from the list.

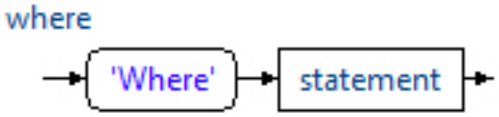


Figure 9-6: Where Syntax Diagram.

9.5 Set

The ‘set’-rule (Figure 9-7) makes it possible to change the value of an attribute. The name of the attribute that needs to be changed is specified by the ‘STRING’-nonterminal. The list of entities which attributes will be changed is referred to by the ‘VARIABLE’-nonterminal. Some attributes are related to properties, while other attributes are related to objects. The ‘set’-rule is only able to change the value of a property, not the object to which an attribute is pointing.

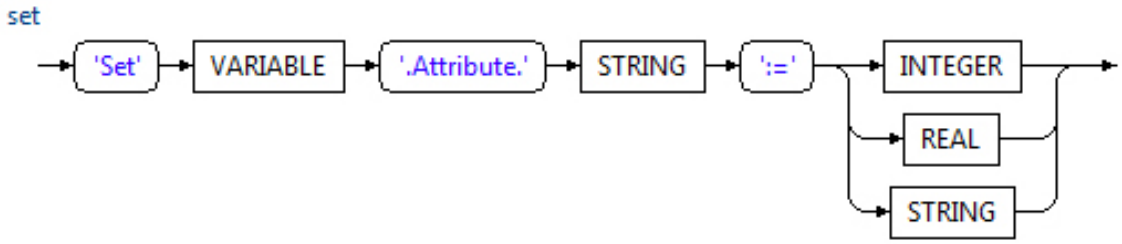


Figure 9-7: Set Syntax Diagram.

9.6 Statement

A statement (Figure 9-8) is a single relation or a combination of several relations. If more relations are specified within one statement these relations are combined using the ‘Or’-token or the ‘And’-token. These tokens indicate a dis- or conjunction between the relations. This single relation or combination of relations is either true or false.

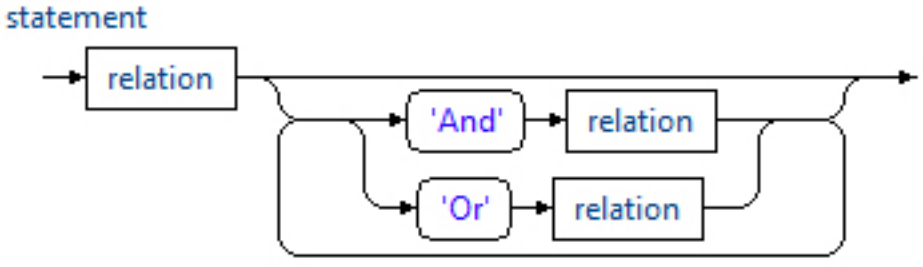


Figure 9-8: Statement Syntax Diagram.

9.7 Relation

The ‘relation’-rule (Figure 9-9) is specified by a ‘relationleft’- and a ‘relationright’-nonterminal and a collection of ‘operator’-tokens that separate them. A relation is either true or false.

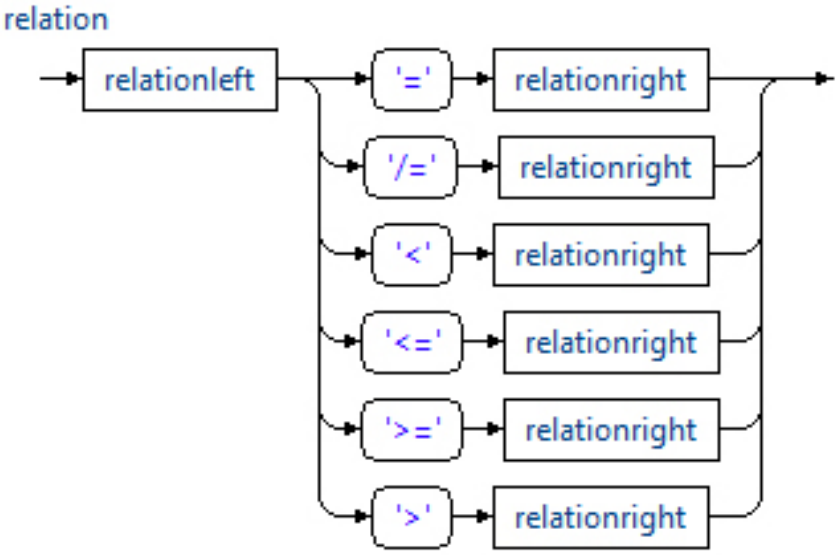


Figure 9-9: Relation Syntax Diagram.

9.8 Relationleft

The ‘relationleft’-rule (Figure 9-10) bears many similarities to the ‘cascade’-rule. While the ‘cascade’-rule was designed to return objects and properties to the end-user, the ‘relationleft’-rule was designed to retrieve properties which can be compared to a value specified by the end-user. This value can be specified using the ‘relationright’-rule.

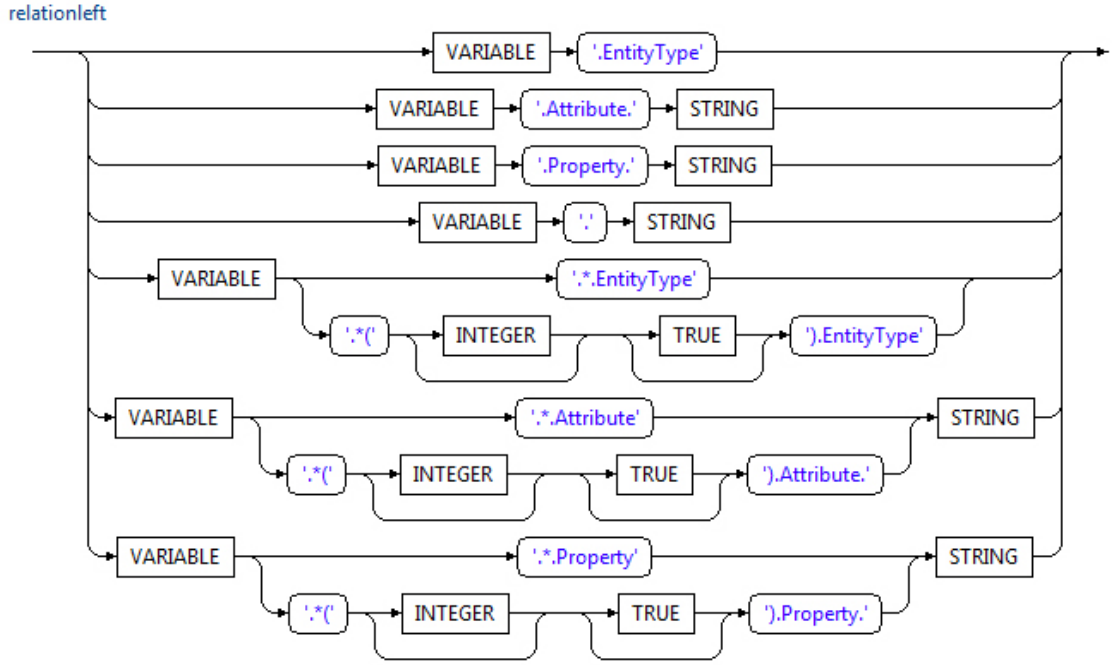


Figure 9-10: Relationleft Syntax Diagram.

The first four options of the ‘relationleft’-rule are similar to options found in the ‘cascade’-rule. They can be used to compare directly related properties to a value specified in the ‘relationright’-rule. The last three options show some similarity with the last option of the ‘cascade’-rule. While the last option of the ‘cascade’-rule returns all directly and indirectly related objects and properties, the last three options of the ‘relationelft’-rule make it possible to compare an the the name of an IFC class and attribute- and property-values of directly and indirectly related objects to a value specified by the end-user. The ‘STRING’-nonterminals are used to specify from exactly which attribute or property the value is needed. The option to specify the depth of the query and the option to ignore ‘IfcRelDefines’-objects, which were described earlier, are also available here.

9.9 Relationright

The ‘relationright’-rule (Figure 9-11) for the assignment of comparison can be a string only, although the figure seems to show otherwise. However if an integer or real is entered it is immediately converted to a string and later when the value it is compared to appears to be numeric, it is cast automatically to an appropriate format. Although the ‘relationright’-rule operates as desired, removing the ‘INTEGER’- and the ‘REAL’-nonterminals improves the BimQL specification, while then these unnecessary items are no longer present. It is also possible to specify patterns by using asterisks, question marks and other regular expression terms. The underlying functionality will try to match the pattern with the value the 'relationleft'-rule returns. These patterns make is possible to return all entities which have a specific attribute, for example 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 specification. Currently our implementation harnesses the built-in regular expressions engine provided by Java. Since different flavors of regular expression engines differ in both expressivity and speed, it might be profitable to investigate alternative implementations to gain speed and versatility in future developments.

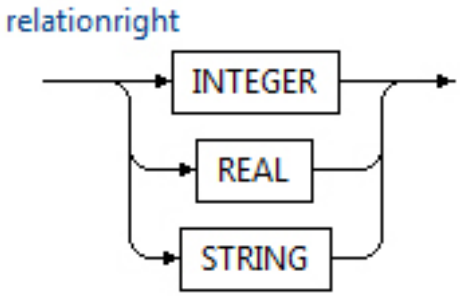


Figure 9-11: Relationright Syntax Diagram.

9.10 Lexer Rules

The previous rules were all parser rules. The next rules however are lexer rules. The lexer rules read characters, divide them into tokens using patterns, and generate a token stream as output. The parser rules read a token stream, generated by the lexer, and match phrases in the language via the rules. Most lexer rules below are quite common and are used in many other grammars. Those common rules are used to identify, strings, numbers and whitespaces.

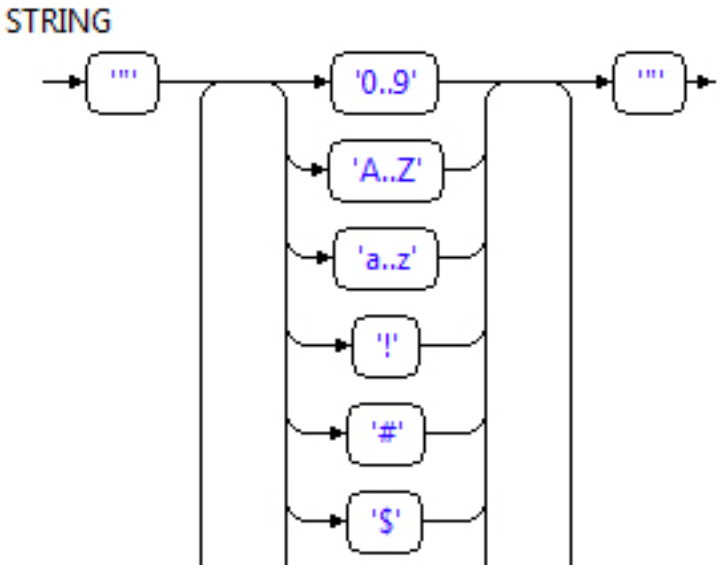


Figure 9-12: String Syntax Diagram.

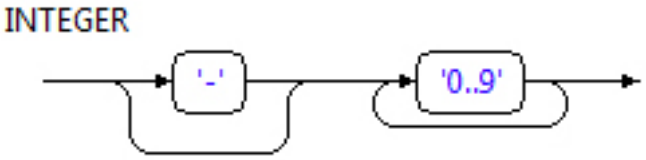


Figure 9-13: Integer Syntax Diagram.

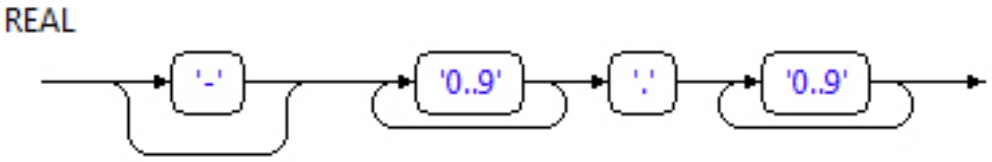


Figure 9-14: Real Syntax Diagram.

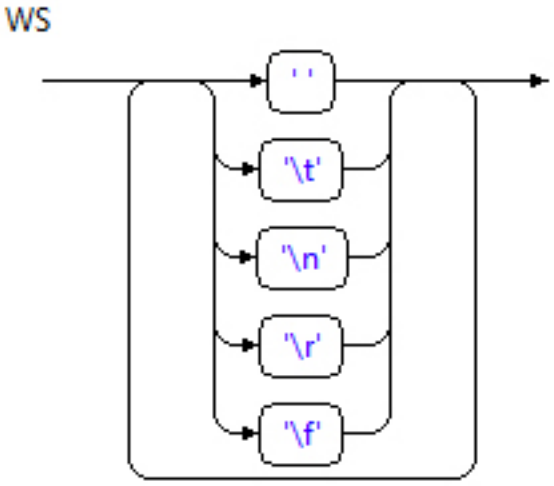


Figure 9-15: Whitespace Syntax Diagram.

A variable (Figure 9-16) is designated by the dollar sign. This syntax is inspired by other languages such as SPARQL.

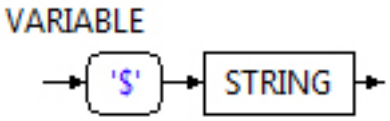


Figure 9-16: Variable Syntax Diagram.

Finally two rules were introduced to give the end-user the possibility to specify true and false (Figure 9-17 and 9-18). Only the ‘true’-token is currently used, however it didn’t take much effort to also create the ‘false’-token.

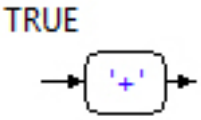


Figure 9-17: True Syntax Diagram.

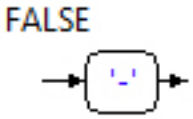


Figure 9-18: False Syntax Diagram.

9.11 Backus–Naur Form

The Backus-Naur Form (BNF) is a notation technique for grammars. Although it might not be as clear as the diagrams above, because it is an exact and textual description, which means no figures are needed, it is often provided when describing programming and other languages (Listing 9-1). The BNF describes the most recent version of the grammar, in which double quotes have been added to the 'DQSTRING'-token and in which the 'DOTSTRING'-token is introduced.

```
1.  bimql ::=
2.      select
3.
4.  select ::=
5.      'Select' VARIABLE where? cascade* set?
6.
7.  cascade ::=
8.      'Select' VARIABLE ':=' VARIABLE
9.      ( | '.EntityType'
10.     | '.Attribute' DOTSTRING
11.     | '.Property' DOTSTRING
12.     | DOTSTRING
13.     | ('.*' | '.*(' INTEGER? TRUE? ')')
14.     ) where?
15.
16. where ::=
17.     'Where' statement
18.
19. set ::=
20.     'Set' VARIABLE '.Attribute' DOTSTRING ':=' (INTEGER | REAL | DQSTRING)
21.
22. statement ::=
23.     relation ('And' relation | 'Or' relation)*
24.
25. relation ::=
26.     relationleft
27.     ('=' relationright
28.     | '/=' relationright
29.     | '<' relationright
30.     | '<=' relationright
31.     | '>=' relationright
32.     | '>' relationright
33.     )
34.
35. relationleft ::=
36.     (VARIABLE '.EntityType'
37.     | VARIABLE '.Attribute' DOTSTRING
38.     | VARIABLE '.Property' DOTSTRING
39.     | VARIABLE DOTSTRING
40.     | VARIABLE ('.*.EntityType' | '.*(' INTEGER? TRUE? ').EntityType')
41.     | VARIABLE ('.*.Attribute' | '.*(' INTEGER? TRUE? ').Attribute') DOTSTRING
42.     | VARIABLE ('.*.Property' | '.*(' INTEGER? TRUE? ').Property') DOTSTRING
43.     )
44.
45. relationright ::=
46.     (INTEGER
47.     | REAL
```

```
48.       | DQSTRING
49.       )
50.
51.   TRUE ::=
52.       '+'
53.
54.   FALSE ::=
55.       '-'
56.
57.   VARIABLE ::=
58.       '$'  ('0..9' | 'A..Z' | 'a..z')+
59.
60.   INTEGER ::=
61.       '-'? '0..9'+
62.
63.   REAL ::=
64.       '-'? '0..9'+ '.' '0..9'+
65.
66.   DQSTRING ::=
67.       '"'
68.       ('0..9' | 'A..Z' | 'a..z'
69.       | '!' | '#' | '$' | '%' | '&' | '^' | '|' | '*' | '+' | ',' | '-' | '.'
70.       | '/' | ':' | ';' | '<' | '=' | '>' | '?' | '~' | '`' | '@' | '_'
71.       )+
72.       '"'
73.
74.   DOTSTRING ::=
75.       '.'
76.       ('0..9' | 'A..Z' | 'a..z'
77.       | '!' | '#' | '$' | '%' | '&' | '^' | '|' | '*' | '+' | ',' | '-' | '.'
78.       | '/' | ':' | ';' | '<' | '=' | '>' | '?' | '~' | '`' | '@' | '_'
79.       )+
80.
81.   WS ::=
82.       (' '
83.       | '\t'
84.       | '\n'
85.       | '\r'
86.       | '\f'
87.       )+
```

Listing 9-1: BNF of BimQL.

10 Integration

Ruben de Laat and Léon van Berlo, both members of the BimServer.org development team, integrated the BimQL framework into the BimServer.org software. This platform already offered two methods for querying building information models, the simple query method and the Java query engine plugin. Now it offers a third, the BimQL method (Figure 10-1).

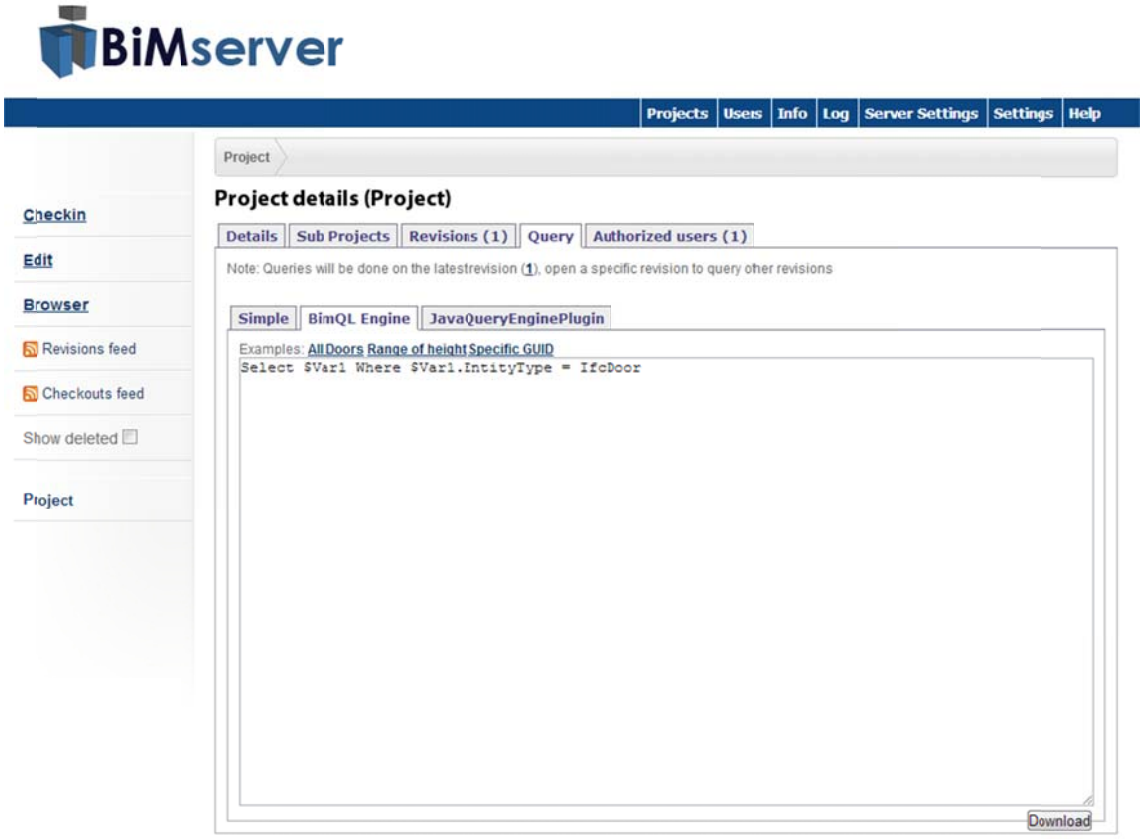


Figure 10-1: BimServer webinterface and BimQL option.

BimServer.org version 1.2.0 beta, released on the 16th of September 2012, was the first release in which BimQL was present and it will probably become one of three standard methods for querying building information models when using the BimServer.org software. It was announced on the 28th of August 2012 on the BimServer.org website. This announcement can be found at http://bimserver.org/tag/bimql/.

Unfortunately it is not yet possible to download a list of strings or numbers, for example a list of Global IDs, a list of dimensions or a list of manufacturers. The BimServer.org development team made the choice to let the software return valid IFC models only. This choice is

very comprehensible, because only then the end-user can be sure the returned model can be opened and edited without any problems. After this issue was brought to the attention of the BimServer.org development team, I was told that this matter would be discussed.

11 Functional Tests

This section reports on some sample queries which were conducted to test if the BimQL design and implementation were functioning as expected.

11.1 Test Description

The tests were performed on the latest grammar which was available on the 24th of November 2012. The IFC model 'AC11-Institute-Var-2-IFC.ifc' was downloaded from the BimServer.org website.

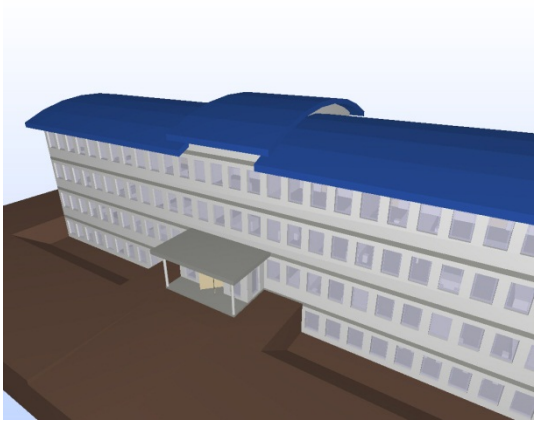
The tests were performed from within the Eclipse development environment and the results were visualized by importing them into the Solibri Model Viewer. The results consisted of a list of Global IDs.

11.2 Test Results

11.2.1 Functional Test 1

The first test (Functional test 11-1) selects the complete model.

1. *Select ?Var1*

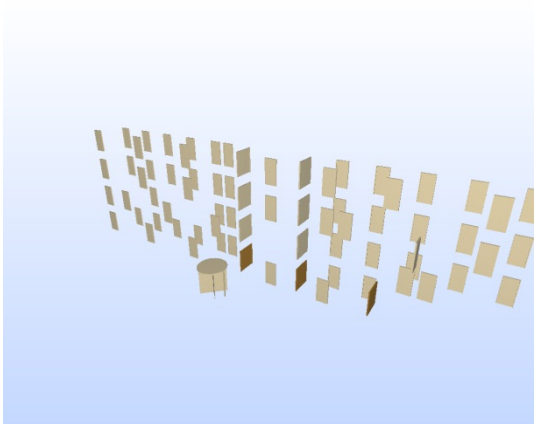


Functional test 11-2: Select everything.

11.2.2 Functional Test 2

The second test (Functional test 11-2) selects all doors from the model.

1. *Select ?Var1*
2. *Where ?Var1.EntityType = "IfcDoor"*

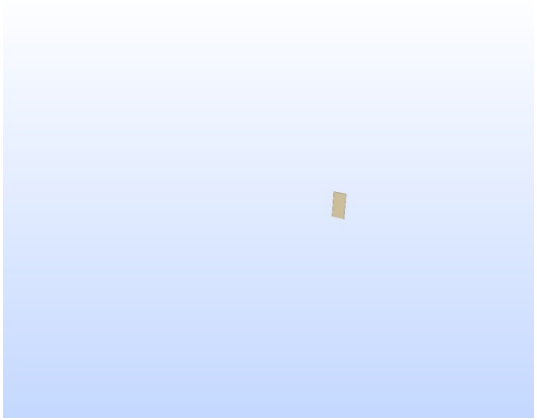


Functional test 11-3: Select all doors.

11.2.3 Functional Test 3

The third test (Functional test 11-3) selects one object. This selection is based on the Global ID of the object.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "32UdM49pfAMxiX5WyXhCuy"*



Functional test 11-4: Base selection on Global ID.

11.2.4 Functional Test 4

Unfortunately the next test (Functional test 11-4) failed. The 'Attribute'-token is still needed to retrieve an attribute.

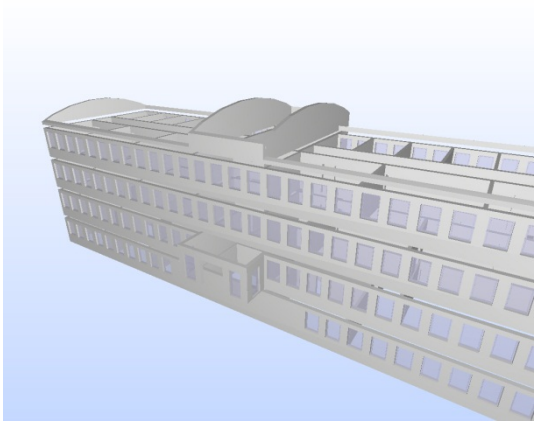
1. *Select ?Var1*
2. *Where ?Var1.GlobalId = "32UdM49pfAMxiX5WyXhCuy"*

Functional test 11-4: Base selection on Global ID, without 'attribute'-token.

11.2.5 Functional Test 5

This query (Functional test 11-5) selects the objects which type-name start with 'IfcW'.

1. *Select ?Var1*
2. *Where ?Var1.EntityType = "IfcW*"*

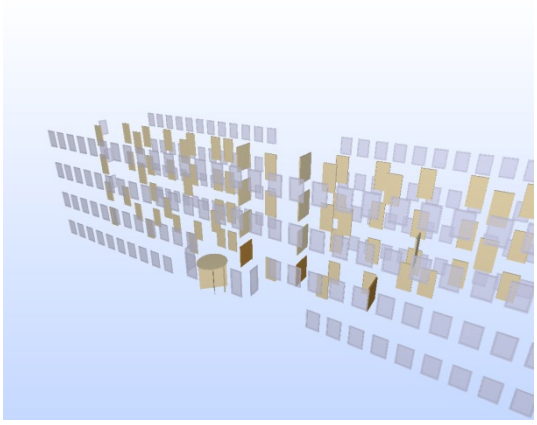


Functional test 11-5: Select object based on entitytype-name.

11.2.6 Functional Test 6

Functional test 6 retrieves all objects which have an attribute with the name OverallHeight

1. *Select ?Var1*
2. *Where ?Var1.Attribute.OverallHeight = "*"*

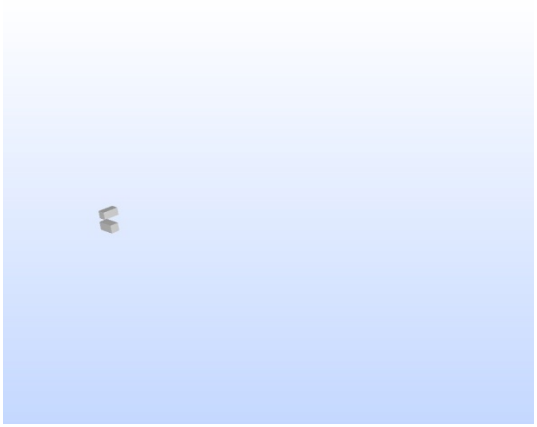


Functional test 11-6: Select objects which have an OverallHeight attribute.

11.2.7 Functional Test 7

Functional test 7 retrieves all red objects. A color is stored in a property-set, it is not an attribute.

1. *Select ?Var1*
2. *Where ?Var1.Property.MainColor = "rot"*



Functional test 11-7: Select all red objects.

11.2.8 Functional Test 8

The next test (Functional Test 11-8) failed. The ‘Property’-token is still needed when querying a property.

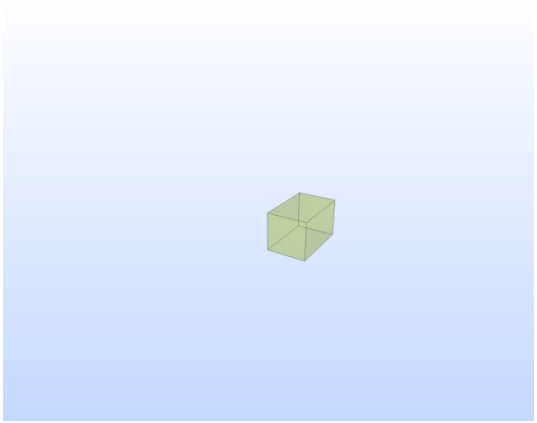
1. *Select ?Var1*
2. *Where ?Var1.MainColor = "rot"*

Functional test 11-8: Select all red objects, without ‘property’-token.

11.2.9 Functional Test 9

Functional test 9 retrieves an object with a specific Global ID.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*

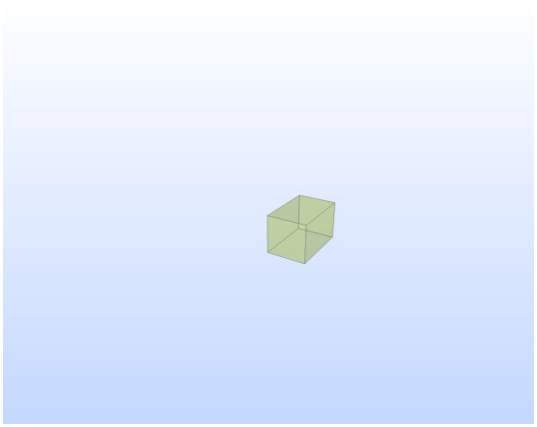


Functional test 11-9: Select object with specific Global ID.

11.2.10 Functional Test 10

Functional test 10 first selects one object, and then it selects all objects 1 deep. Although these objects are not visible, this test was successful.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*
3. *Select ?Var2 := ?Var1.**

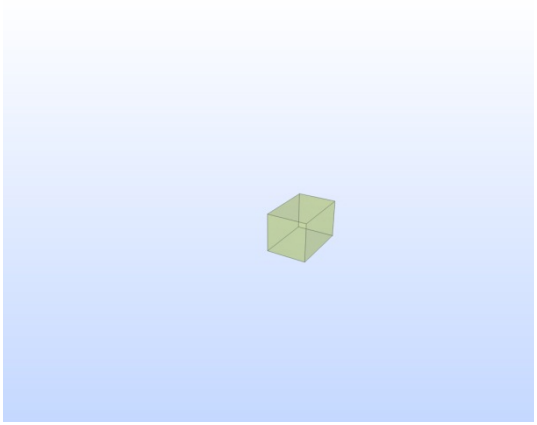


Functional test 11-10: Select related objects, 1 deep.

11.2.11 Functional Test 11

Functional test 11 first selects one object, and then it selects all related objects, again 1 deep. Although these objects are not visible, this test was successful.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*
3. *Select ?Var2 := ?Var1.*(1)*

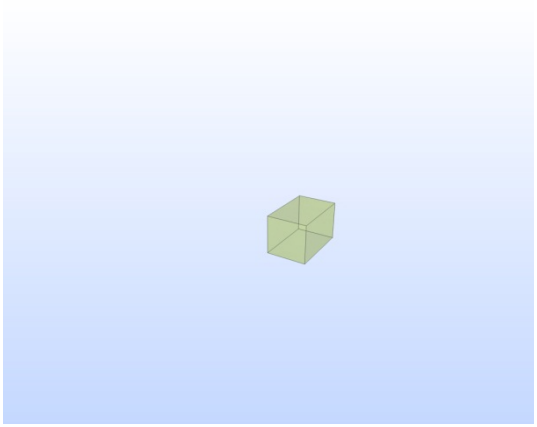


Functional test 11-11: Select related objects, again 1 deep.

11.2.12 Functional Test 12

Functional test 12 first selects one object, then it selects all objects 2 deep. Although these objects are not visible, this test was successful.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*
3. *Select ?Var2 := ?Var1.*(2)*

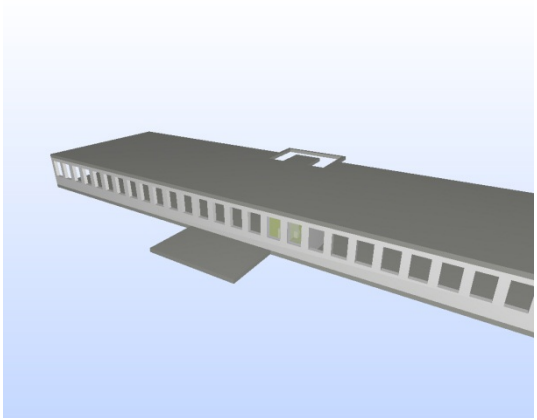


Functional test 11-12: Select related objects, 2 deep.

11.2.13 Functional Test 13

Functional test 13 first selects one object, and then it selects all objects 3 deep.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*
3. *Select ?Var2 := ?Var1.*(3)*

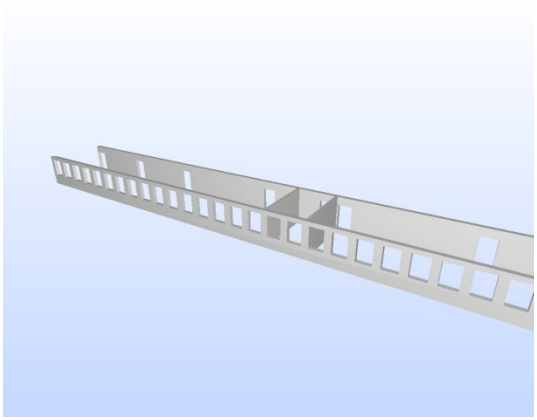


Functional test 11-13 Select related objects, 3 deep.

11.2.14 Functional Test 14

Functional test 14 retrieves all walls related to a specific space.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*
3. *Select ?Var2 := ?Var1.*(2)*
4. *Where ?Var2.EntityType = "IfcWallStandardCase"*

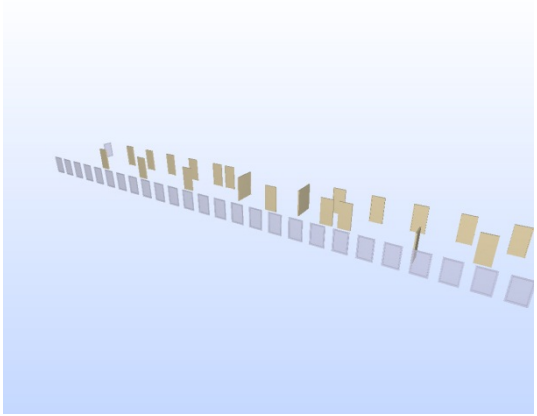


Functional test 11-14: Select related walls.

11.2.15 Functional Test 15

Functional test 15 retrieves all doors and windows related to the walls selected in functional test 14.

1. *Select ?Var1*
2. *Where ?Var1.Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*
3. *Select ?Var2 := ?Var1.*(2) Where ?Var2.EntityType = "IfcWallStandardCase"*
4. *Select ?Var3 := ?Var2.*(4) Where ?Var3.EntityType = "IfcDoor" Or ?Var3.EntityType = "IfcWindow"*

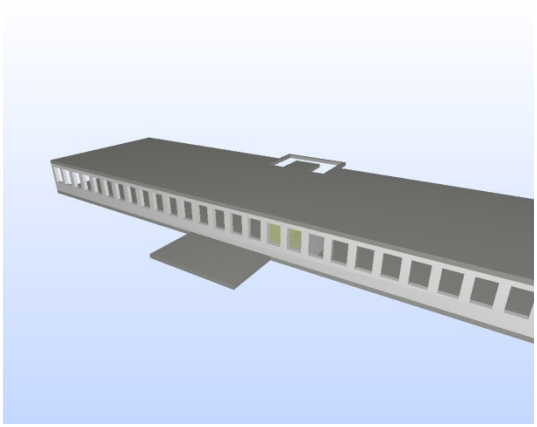


Functional test 11-15: Select related doors and windows.

11.2.16 Functional Test 16

Functional test 16 shows antoher method to select all objects related to one specific object.

1. *Select ?Var1*
2. *Where ?Var1.*(2).Attribute.GlobalId = "3iSzpa9d93jhTDB7hG0QBW"*



Functional test 11-16: Select related objects.

11.3 Test Conclusion

Most test results were on par with our expectations. Unfortunately the feature which discharges the end-user from deciding whether a property or an attribute is involved is not (yet) functioning. This has probably something to do with the grammar, however I expect to solve this issue shortly.

12 Performance Tests

In order to test performance, we have conducted a number of tests using different models and queries.

12.1 Test Description

All tests have been carried out on standard contemporary year 2012 hardware (Intel Core 2 Duo P8600 processor, which runs at 2.4 GHz, 4 GB main memory and a 300 GB, 7200 rpm hard disk). BimQL addresses BimServer.org functionality and therefore most of the time it is BimServer.org functionality running instead of BimQL functionality. However other developers might change how BimQL addresses BimServer.org functionality and if performance is an issue the performance of that new method can be compared with the performance of current methods.

BimQL was integrated into BimServer.org version 1.2.0 beta. Unfortunately we could not use this version to test our domain specific language, because we were unable to communicate with the BimServer through its SOAP interface. The SOAP interface can be used to automate the interaction with the BimServer. So instead of performing multiple queries manually, code can be written to perform those queries automatically one after another. After we mentioned this problem on the BimServer.org support forum, the problems present in the BimServer.org software were quickly solved by the BimServer.org development team. They even provided a script, which automatically carries out BimQL queries. So eventually the performance tests were run on the trunk version downloaded on the 16th of October 2012 from http://bimserver. googlecode.com/svn/trunk.

The tests documented here use three different models that are available from the Open IFC Model Repository. Three exemplary queries where issued repeatedly through the SOAP interface to the BimServer.org system running on the local host. Query 1: 'Select $Var1', selecting all entity instances from the model. Query 2: 'Select $Var1 Where $Var1.Attribute.GlobalId = [Global ID from model]', selecting a single entity from the respective model. Query 3: 'Select $Var1 Where $Var1.EntityType = IfcDoor', selecting all instances of IfcDoor existing in the respective models.

12.2 Test Results and Conclusion

	# of entities	Query 1: time (sec)	Query 1: # of results	Query 2: time (sec)	Query 2: # of results	Query 3: time (sec)	Query 3: # of results
Duplex Apartment Model	38906	3.455	3893	3.226	1	3.094	14
AC11-Institute-Var-2-IFC	49674	3.602	6565	3.277	1	2.796	77
AC-11-Smiley-West-04-07-2007	73665	4.484	2565	3.552	1	4.000	85

Table 1: Overview of the models and queries tested.

The results depicted in table 1 of the queries show a linear increase of execution time in relation with the number of entities a model consists of. The repeated execution of queries showed variations of execution times in insignificant ranges of a few milliseconds, which allows drawing the conclusion that neither positive nor negative side-effects such as caching are to be expected in practice.

13 Conclusion and Future Work

In this report BimQL was described. BimQL is an open domain specific query language for building information models. It can be used to select and update partial aspects in building information models. This report described the steps taken during the design process of the language.

After the requirements were specified and related research performed by others was examined, the BimQL specification was developed. The language was designed and implemented on top of the BimServer.org platform, however while the specification of BimQL is based on the IFC model specification, BimQL can be integrated in any other IFC based modeling or development tool.

One of the requirements was to adhere to the CRUD-principle. During this project we focused mainly on the read- and select-part of that principle and we also implemented the functionality to update or set the value of an attribute. The create- and the delete-part of the CRUD-principle were not implemented. These operations are more complicated and therefore more research is required.

Another requirement was streamlining the retrieval of properties and directly and indirectly related objects. To meet that requirement the '.property'- and '.*'-token were introduced, which were successfully implemented. The design provides the option to omit the '.property'- and the '.attribute'-token. The implementation of this option was started, however is not yet functional.

We received positive feedback and hope other enthusiasts will further develop and improve our domain specific language. To facilitate that, we made the source code available through GitHub and we will document the language at bimql.org.

Future development may focus on natural language enhancements that would potentially increase usability by allowing end-users to operate with non-technical vocabulary ('Walls', 'Wall', 'Wand', 'Muur' would then cover 'IfcWall' and 'IfcWallStandardcase').

Other syntactic shortcuts, similar to the ‘property’-operator, might also be developed. Those shortcuts would ease the creation of simple and complex queries by predefining often used partial queries. A shortcut that simplifies searching for certain specific dependencies between entities could be created. This would make it easier to for example find all doors or windows related to one wall. This is already possible now, however more than one line of query code is needed.

Further future potential extensions of the work include the introduction of spatial queries, providing the possibility to retrieve information about geometric and topologic entities in a building information model. It would then for example be possible to search for all south faced windows and all outside doors on the ground floor. While developing the spatial query functionality, simultaneously new shortcuts could be developed to accelerate the development of queries.

Even though this query language is far from feature complete, BimQL can be a useful vantage point for future research, discussion and development.

14 Acknowledgements

Although the development of this framework could probably last forever, my journey ends her. The last year has been a good experience. The design and development of a computer language was something new to me which I enjoyed discovering. I had the impression other members of the BIM-community were also interested in this project. That and the prospect of this framework being integrated into the BimServer.org-platform were the drive trying to make this project a success.

At the start of this project the expectations were high and we had many ideas in store. We talked about graphical user interfaces, spatial queries and probably much more I cannot remember now. Although we did not implement most ideas, a clear and stable framework has been created which can be taken further by anyone who is interested in what we did.

In July 2012 I went to the EG-ICE workshop which was organized by the Technical University Munich. At this workshop I presented the paper which described the development and the current status of the BimQL project. That paper was written together with Jakob Beetz, my supervisor. I want to thank him for giving me the opportunity to present the paper at that event and for the supporting me during this project. The paper can be found at the end of this document.

I would like to extend my sincere gratitude and appreciation to the bimserver.org community and especially to Ruben de Laat and Léon van Berlo of TNO and to Joran Jessurun of the Eindhoven University of Technology for their support and feedback during this work.

15 References

1. 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.

2. 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.

3. 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.

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

5. 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).

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

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

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

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

10. H. Kang, G. Lee, Development of an Object-Relational IFC Server, in: Proc. of the 3rd International Conference on Construction Engineering & Management / 6th International Conference on Construction Project Management, Jeju, Korea, 2009.

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

12. 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).

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

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

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

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

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

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

19. Weise, M., Katranuschkov, P., Scherer, R. (2003) Generalised Model Subset Definition schema. In Construction

20. D. Koonce, L. Huang, R. Judd, EQL an Express Query Language, Computers & Industrial Engineering. 35 (1998) 271–274.

16 Appendix A: Source Code

The implementation of the design starts with the conversion from the Backus Naur Form (BNF) notation into an ANTLR grammar. This grammar consists of a lexer and a parser, which can read an input stream and divide it into tokens. The lexer and parser can recognize BimQL, however a grammar alone is not capable of performing any actions.

The BimServer.org project provides the possibility to create complex and extensive queries by writing Java code. Consequently all functionality desired is already present, however the actual problem is addressing this functionality. The development of the BimQL framework solves this problem.

The most obvious next step might be directly connecting the newly developed grammar and the already existing functionality present within the BimServer.org project. However this is not how it was done, because it would be too complicated to make the translation from the grammar to the BimServer.org functionality within the grammar. Instead a number of transition Java classes were written first. One of these Java classes is for example able to search for an attribute or property and another one ANDs two lists of Booleans. This means the BimServer.org functionality is not directly addressed by the grammar. Instead it is addressed by the transition classes which are the ones actually being addressed by the grammar.

In this part the ANTLR grammar is discussed first. Then the generation and operation of the transition classes is explained and finally the relation between the two is clarified.

16.1 ANTLR Grammar

The ANTLR grammar (Listing 16-1) looks very similar to the Backus Naur Form (BNF) notation. Notice the action at the end of the 'whitespace'-rule, rule 16. This action places the 'whitespace'-tokens on a hidden channel. They are still sent to the parser, but are invisible. Whitespaces will be ignored and therefore the character streams 'Select $Var1' and 'Select$Var1' will be treated equally by the parser.

```
1.  bimql: select;
2.  select: 'Select' VARIABLE where? cascade* set?;
3.  cascade: 'Select' VARIABLE' := 'VARIABLE ( | '.EntityType' | '.Attribute.'STRING |
    '.Property.' STRING | '.' STRING | (('.*') | ('.*('INTEGER? TRUE? ')' ))) (where)?;
4.  where: 'Where' statement;
5.  set: 'Set' VARIABLE '.Attribute.' STRING ':=' (INTEGER | REAL | STRING);
6.  statement: relation1=relation ('And' relation | 'Or' relation)*;
7.  relationreturns: relationleft ('=' relationright | '/=' relationright | '<' relationright | '<='
    relationright | '>=' relationright | '>' relationright);
8.  relationleft: VARIABLE '.EntityType' | VARIABLE '.Attribute.' STRING | VARIABLE
    '.Property.' STRING | VARIABLE '.' STRING | VARIABLE ('.*.EntityType' | '.*(' INTEGER?
    TRUE? ').EntityType') | VARIABLE ('.*.Attribute' | '.*(' INTEGER? TRUE? ').Attribute.')
    STRING | VARIABLE ('.*.Property' | '.*(' INTEGER? TRUE? ').Property.') STRING;
9.  relationright: INTEGER | REAL | STRING;
10. TRUE: '+';
11. FALSE: '-';
12. VARIABLE: '$' STRING;
13. INTEGER: ('0'..'9')+;
14. REAL: INTEGER+ ('.' INTEGER+)?;
15. STRING: ('\u002A'..'\u002A' | '\u0024'..'\u0024' | '\u003F'..'\u003F' | '\u0030'..'\u0039' |
    '\u0041'..'\u005A' | '\u005B'..'\u0060' | '\u0061'..'\u007A')+;
16. WS: (' ' | '\t' | '\n' | '\r' | '\f')+ {$channel=HIDDEN;};
```

Listing 16-1: ANTLR grammar.

16.2 Transition Classes

Each class described in this section performs a specific action which can be initiated by the parser. Each transition class and probably each other Java class can be divided into a few sections. During this project and maybe in general the most important sections are the field-, the constructor- and the method-section. The field-section defines the class-variables, the constructor-section contains some special methods which can instantiate the class and all the other methods are grouped together in the method-section. These methods can for example alter variables defined earlier or return the solution of a calculation.

16.2.1 'GetAttribute'-Classes

The 'GetAttribute'-classes make it possible to retrieve the value of an attribute. The height of a door for example is stored in the attribute with the name 'OverallHeight'. All the classes which are necessary to extract information from attributes are located in the package named 'nl.wietmazairac.bimql.get.attribute'. One of those classes is called

‘GetAttributeMain.java’ and that class is the starting point when retrieving an attribute. Each single other class is related to one IFC entity type. For example the class related to the entitytype IfcDoor is called ‘GetAttributeSubIfcDoor’.

16.2.1.1 ‘GetAttributeMain’-Class

When the value of an attribute needs to be retrieved, the ‘GetAttributeMain’-class is the first class to be called. This class determines the entitytype of the object of which the value of an attribute needs to be retrieved. Subsequently the ‘GetAttributeMain’-class calls the appropriate ‘GetAttributeSub’-class, which actually retrieves the value of an attribute.

To operate correctly the ‘GetAttributeMain’-class requires a list of objects and a string, a sequence of characters. The string describes the name of the attribute which value needs to be retrieved. The objects contained by the list are the objects from which the value of one of its attributes might need to be fetched. The constructor of this class is shown in listing 16-2.

```
1. public GetAttributeMain(List<Object> ObjectList, String string) {
2. this.ObjectList = ObjectList;
3. this.string = string;
4. }
```

Listing 16-2: GetAttributeMain-constructor.

For each object in the list the ‘GetAttributeMain’-class determines its type and it then creates an instance of the appropriate ‘GetAttributeSub’-class. Part of this loop is shown in listing 16-3. The instance of the ‘GetAttributeSub’-class returns the value of the attribute which is stored in a list.

```
1.  for (Object object : ObjectList) {
2.  if (((IfcRoot) object).eClass().getName().equals("Ifc2DCompositeCurve")) {
3.      GetAttributeSubIfc2DCompositeCurve GetAttributeSubIfc2DCompositeCurve = new
    GetAttributeSubIfc2DCompositeCurve(object, string);
4.      objectArrayList = GetAttributeSubIfc2DCompositeCurve.getResult();
5.      arrayListArrayList.add(objectArrayList);
6.  }
7.  ...
8.  ...
9.  ...
10. else if (((IfcRoot) object).eClass().getName().equals("IfcDoor")) {
11.     GetAttributeSubIfcDoor GetAttributeSubIfcDoor = new GetAttributeSubIfcDoor(object,
    string);
12.     objectArrayList = GetAttributeSubIfcDoor.getResult();
13.     arrayListArrayList.add(objectArrayList);
14. }
15. ...
16. ...
17. ...
18. else {
19. }
20. }
```

Listing 16-3: GetAttributeMain-class.

16.2.1.2 ‘GetAttributeSub’-Class

A ‘GetAttributeSub’-class is called by the ‘GetAttributeMain’-class after the entitytype of the object from which the value of an attribute needs to be retrieved has been determined. The ‘GetAttributeSub’-classes have been created to solve typecasting problems. Keeping the code organized was another benefit. Not all entitytypes own the same attribute types and by taking this route that has been taken into account.

A ‘GetAttributeSub’-class needs an object and a string. It will check if that object has an attribute with the same name as the string. The constructor of this class is shown in listing 16-4.

```
1.  public GetAttributeSubIfcDoor(Object object, String string) {
2.  this.object = object;
3.  this.string = string;
4.  }
```

Listing 16-4: GetAttributeSubIfcDoor-constructor.

The ‘GetAttributeSub’-class tries to match the provided string with the name of an attribute. Part of that code is shown in listing 16-5. If a match occurs, BimServer.org functionality is executed to retrieve the required value. This value is then stored in a list. It is possible that the value of an attribute is not one object or value, but a list of objects or values. This has been accounted for as shown in the code below. The ‘OverallWidth’-section only adds one value or object to the list called ‘resultList’, wether the ‘ConnectedTo’-section might add more than one if the size of the retrieved list (“getConnectedTo().size()”) is larger than one.

```
1.  if (string.equals("OverallWidthAsString")) {
2.  resultList.add(((IfcDoor) object).getOverallWidthAsString());
3.  }
4.  else if (string.equals("OverallWidth")) {
5.  resultList.add(((IfcDoor) object).getOverallWidth());
6.  }
7.  ...
8.  ...
9.  ...
10. else if (string.equals("FillsVoids")) {
11. for (int i = 0; i < ((IfcDoor) object).getFillsVoids().size(); i++) {
12.     resultList.add(((IfcDoor) object).getFillsVoids().get(i));
13. }
14. }
15. else if (string.equals("ConnectedTo")) {
16. for (int i = 0; i < ((IfcDoor) object).getConnectedTo().size(); i++) {
17.     resultList.add(((IfcDoor) object).getConnectedTo().get(i));
18. }
19. }
```

Listing 16-5: GetAttributeSubIfcDoor-class.

16.2.1.3 Generation of 'GetAttribute'-Classes

Many 'GetAttribute'-classes are present in the package named 'nl.wietmazairac.bimql.get.attribute'. To all write them manually would have been a cumbersome task and therefore it was decided to apply the concept of metaprogramming. While metaprogramming, the programmer does not write the final code itself, instead he writes the code that generates that final code for him. This concept may be applied when the final code is very long and when that code contains a lot of repetitive elements, like in our case.

The class that generates the 'GetAttribute'-classes is called 'CreateGetAttributeMainSubObjectWrap.java' and is located in the package named 'nl.wietmazairac.bimql'. This class first loops through all the classes located in the package named 'org.bimserver.models.ifc2x3.impl'. If the name of a class starts with "Ifc" and ends with "Impl", a second loop is initiated. This second loop checks whether the class found in the first loop has any methods that start with "get". The methods the second loop goes through are part of the BimServer.org functionality. A method that starts with "get" is, in this particular case, a method that retrieves the value of an attribute. Now the classes of interest and the relevant methods are known, the 'java.io.Printwriter'-class can be used to generate the 'GetAttribute'-classes.

There was another issue that needed to be addressed and it was designated the 'WrappedValue'-problem. This problem revealed itself when trying to retrieve all GlobalIDs of a list of objects. Instead of retrieving a list of strings, a list of objects was returned. This is what actually should happen, but it is probably not something a non-expert user would expect. Therefore another if clause was added which stated that if a retrieved objects owns a method called 'getWrappedValue' , not the object itself should be retrieved, however the value wrapped should be instead. It is still possible to retrieve the object instead of the wrapped value. This is done by appending the character string 'object'. For example to fetch the 'GlobalID'-object instead of the 'GlobalID'-string, 'GlobalIdObject' should be entered instead of 'GlobalId'. This was also demonstrated earlier in the examples.

16.2.2 'SetAttribute'-Classes

The 'SetAttribute'-classes make it possible to change the value of an attribute. All the classes which are necessary to change the values stored in attributes are located in the package named 'nl.wietmazairac.bimql.set.attribute'.

The 'SetAttribute'-classes and the 'GetAttribute'-classes are very similar. The way the 'GetAttribute'-classes operate and the way they were created was described before and the same methods apply to the 'SetAttribute'-classes.

16.2.3 'GetProperty'-Class

The 'GetProperty'-class makes it possible to retrieve the value of a property. Properties are different from direct attributes. An almost unlimited amount of properties can be added to an object. This contrasts with the small amount of attributes already defined in the core schema. Another difference is the amount of network hops that need to be taken before the required property is reached. Until the deployment of BimQL, when users were still required to write advanced queries in Java, it was more complicated to retrieve a property than it was to retrieve an attribute. This class eliminates that problem. The 'GetProperty'-class is located in the package named 'nl.wietmazairac.bimql.get.property'.

16.2.3.1 'GetPropertyMain'-Class

The 'GetPropertyMain'-class needs a list of objects and a string to function. This class will check of any of those objects owns a property which name is equal to the string. If so, the value of that property will be stored in a list. The constructor of this class is shown in listing 16-6.

```
1.  public GetPropertyMain(List<Object> objectList, String string) {
2.      this.objectList = objectList;
3.      this.string = string;
4.  }
```

Listing 16-6: GetPropertyMain-constructor.

The hops that need to be taken while retrieving a property are shown in figure 8-1 and It explains the code in listing 16-7. Line 1 defines the start of a loop. This means it executes the lines within the curly brackets for each object in the list. That object is equivalent to the 'IfcSpace'-object in figure 4. It could be a space, however it could as well be any other object. Line 3 checks if the object is an instance of an 'IfcObject'. If not, it skips the current object, otherwise an error could occur later on. Line 4, 5 and 6 fetch all objects related to the

current object and it checks which of those objects have a name equal to ‘IfcRelDefinesByProperties’. The objects that meet this requirement will be processed further. Line 7 and 8 get the relating property definition and check if that property definition is of the type ‘IfcPropertySet’. If that’s true the properties of that set are added to a temporary list called ‘ifcPropertyList’. This happens in line 9. In line 10 a loop is defined which checks every object contained by this new list. If the name of a property from the ‘ifcPropertyList’ is equal to the string encountered earlier, the value of that property needs to be retrieved. Line 12 and line 29 determine if the property which value needs to be fetched, is a complex property or a single value property. A complex property is a collection of single value properties. Color is a complex property for example. Each color is defined by a red, a green and a blue value (RGB) or by a hue, a saturation and a brightness value (HSB). The three single value properties are stored together in one complex property. Finally the data type stored in a property needs to be determined. This is done line 13 through 26 and in line 32 through 45. Once the data type has been determined the value can be stored into a temporary list. This happens in line 14, line 20, line 26, line 33, line 39 and in line 45.

```
1.  for (Object object : objectList) {
2.    ArrayList<Object> objectArrayList = new ArrayList<Object>();
3.    if (object instanceof IfcObject) {
4.      List<IfcRelDefines> ifcRelDefinesList = new ArrayList<IfcRelDefines>(((IfcObject)
    object).getIsDefinedBy());
5.      for (IfcRelDefines ifcRelDefines : ifcRelDefinesList) {
6.        if (ifcRelDefines.eClass().getName().equals("IfcRelDefinesByProperties")) {
7.          IfcPropertySetDefinition ifcPropertySetDefinition = ((IfcRelDefinesByProperties)
    ifcRelDefines).getRelatingPropertyDefinition();
8.          if (ifcPropertySetDefinition.eClass().getName().equals("IfcPropertySet")) {
9.            List<IfcProperty> ifcPropertyList = new ArrayList<IfcProperty>(((IfcPropertySet)
    ifcPropertySetDefinition).getHasProperties());
10.           for (IfcProperty ifcProperty : ifcPropertyList) {
11.             if (ifcProperty.getName().equals(string)) {
12.               if (ifcProperty.getClass().getSimpleName().equals("IfcPropertySingleValueImpl")) {
13.                 if (((IfcPropertySingleValue)
    ifcProperty).getNominalValue().getClass().getSimpleName().equals("Ifc2x3Package")) {
14.                   objectArrayList.add(((Ifc2x3Package) ((IfcPropertySingleValue)
    ifcProperty).getNominalValue()).getWrappedValue());
15.                 }
16.                 ...
17.                 ...
18.                 ...
19.                 else if (((IfcPropertySingleValue)
    ifcProperty).getNominalValue().getClass().getSimpleName().equals("IfcInteger")) {
```

```
20.                 objectArrayList.add(((IfcInteger) ((IfcPropertySingleValue)
    ifcProperty).getNominalValue()).getWrappedValue());
21.               }
22.               ...
23.               ...
24.               ...
25.               else if (((IfcPropertySingleValue)
    ifcProperty).getNominalValue().getClass().getSimpleName().equals("IfcReal")) {
26.                 objectArrayList.add(((IfcReal) ((IfcPropertySingleValue)
    ifcProperty).getNominalValue()).getWrappedValue());
27.               }
28.             }
29.             else if (ifcProperty.getClass().getSimpleName().equals("IfcComplexPropertyImpl"))
    {
30.               List<IfcProperty> ifcComplexPropertyList = new
    ArrayList<IfcProperty>(((IfcComplexProperty) ifcProperty).getHasProperties());
31.               for (IfcProperty ifcComplexProperty : ifcComplexPropertyList) {
32.                 if (((IfcPropertySingleValue)
    ifcComplexProperty).getNominalValue().getClass().getSimpleName().equals("Ifc2x3Package
    ")) {
33.                   objectArrayList.add(((Ifc2x3Package) ((IfcPropertySingleValue)
    ifcComplexProperty).getNominalValue()).getWrappedValue());
34.                 ...
35.                 ...
36.                 ...
37.                 }
38.                 else if (((IfcPropertySingleValue)
    ifcComplexProperty).getNominalValue().getClass().getSimpleName().equals("IfcInteger")) {
39.                   objectArrayList.add(((IfcInteger) ((IfcPropertySingleValue)
    ifcComplexProperty).getNominalValue()).getWrappedValue());
40.                 }
41.                 ...
42.                 ...
43.                 ...
44.                 else if (((IfcPropertySingleValue)
    ifcComplexProperty).getNominalValue().getClass().getSimpleName().equals("IfcReal")) {
45.                   objectArrayList.add(((IfcReal) ((IfcPropertySingleValue)
    ifcComplexProperty).getNominalValue()).getWrappedValue());
46.                 }
47.               }
48.             }
49.           }
50.         }
51.       }
52.     }
53.   }
54. }
```

Listing 16-7: GetPropertyMain-class.

16.2.3.2 Generation of 'GetProperty'-Classes

Just like the 'GetAttribute'- and 'SetAttribute'-classes, the 'GetPropertyMain'-class could not be written manually. Many different data types need to be processed by this class and they all had to be specified to prevent casting problems. Listing 16-8 shows the essence of the 'CreateGetPropertyMain'-class. Line 5 checks if a class owns a method which is named 'getWrappedValue'. If that is true line 8 through 15 add the lines to the 'GetPropertyMain'-class which check if the current property is the property being searched for and the lines, which cast that property to the appropriate type.

```
1.  int j = 0;
2.  for (int i = 0; i < ClassArray.length; i++) {
3.    Method method[] = ClassArray[i].getDeclaredMethods();
4.    for (Method aMethod : method) {
5.      if (aMethod.getName().equals("getWrappedValue")) {
6.        System.out.println(i);
7.        if (j == 0) {
8.          printWriter.print("              if ");
9.        } else {
10.         printWriter.print("              else if ");
11.       }
12.       printWriter.println("(((IfcPropertySingleValue)
      ifcProperty).getNominalValue().getClass().getSimpleName().equals(\"" +
      ClassArray[i].getSimpleName() + "\")) {");
13.       printWriter.println("                objectArrayList.add(((" +
      ClassArray[i].getSimpleName() + ") ((IfcPropertySingleValue)
      ifcProperty).getNominalValue()).getWrappedValue());");
14.       printWriter.println("              }");
15.       j++;
16.     }
17.   }
18. }
```

Listing 16-8: CreateGetPropertyMain-class.

16.2.4 'GetEntityType'-class

The 'GetEntityType'-class makes it possible to determine what the type of an object is, for example 'IfcDoor' or 'IfcWall'. This class is located in its own package named 'nl.wietmazairac.bimql.get.entitype'. It needs a list of objects to operate and it returns a two dimensional list of strings. A two dimensional list of strings could also be described as a list of lists of strings. This class does not require a two dimensional list as one object can only be of one type. However there are cases in which this two dimensional construction is actually indispensable. A wall for example can be connected to more than one other wall. In this case the 'first dimension'-list contains all the walls. Only one 'first-dimension'-list exists. The amount of 'second dimension'-lists is equals

to the amount of walls contained by the ‘first dimension’-list. The first ‘second dimension’-list contains the walls which are connected to the first wall stored in the ‘first dimension’-list. The second ‘second-dimension’-list contains the walls which are connected to the second wall stored in the ‘first dimension’-list, and so on.

The reason for using this two dimensional construction here, although not strictly necessary, is the way the others classes operate. The constructors of for example the ‘relational operators’-classes, which will be discussed later, are expecting a two dimensional list. Another option might have been to let the ‘GetEntityType’-class generate a ‘one dimensional’-list and create new constructors for the ‘relational operators’-classes. This however was not the preferred route, because then additional code was generated to handle exceptions while it is often better to fit exceptions into the big picture.

16.2.5 Relational Operators

Six relational operators have been implemented. Present are the equal-, the in-equal, the less-, the less-equal-, the greater- and the greater-equal-operator. Only the equal-operator will be discussed here, because it is the most complicated relational operator and because the other relational operators are very similar.

16.2.5.1 ‘EqualOperator’-Class

The relational operators compare a two dimensional list, a list of lists of doubles or strings to a string. It returns a list of Booleans. The constructor of this class is shown in listing 16-9.

```
1.  public EqualOperator(List<ArrayList> leftOperand, String rightOperand) {
2.      this.leftOperand = leftOperand;
3.      this.rightOperand = rightOperand;
4.  }
```

Listing 16-9: EqualOperator-constructor.

The ‘EqualOperator’-class (listing 16-10) first checks whether the left operand is a string or a double. This happens in line 5 and in line 20. When the left operand is a string, regular expressions can be used to compare the left and the right operand. A regular expression is a pattern that can be matched to an input character stream. For example the pattern ‘Floor*’ will match the input text ‘Floor2’, ‘Floor13’ and ‘Floor4a’. The pattern ‘Room???’ will match the input stream ‘Room429’, ‘Room5w3’ and ‘Room78e’. When the left operand is a double the right operand will be parsed to a double and then they will be compared. However if the right operand is an asterisk symbol, any

double will be matched. The asterisk symbol is commonly used as a wildcard character.

If the right operand is a string, it will first be converted to a regular expression pattern. The ‘EqualOperator’-class tries to match the left operand to this pattern. If a match occurs ‘true’ is added to the list of results. If no match has occurred a ‘false’ will be added.

```
1.  for (ArrayList arrayList : leftOperand) {
2.      if (arrayList.size() > 0) {
3.        for (int i = 0; i < arrayList.size(); i++) {
4.          if (arrayList.get(i) != null) {
5.            if (arrayList.get(i).getClass().getSimpleName().equals("Double")) {
6.              if (rightOperand.indexOf("*") < 0) {
7.                double rightOperandDouble = Double.parseDouble(rightOperand);
8.                if (arrayList.get(i).equals(rightOperandDouble)) {
9.                  result.add(true);
10.                 break;
11.               } else {
12.                 if (i == arrayList.size() - 1) {
13.                   result.add(false);
14.                 }
15.               }
16.             } else if (rightOperand.equals("*")) {
17.               result.add(true);
18.               break;
19.             }
20.           } else if (arrayList.get(i).getClass().getSimpleName().equals("String")) {
21.             String regex = rightOperand;
22.             regex = regex.replace(".", "\\.");
23.             regex = regex.replace("*", ".*");
24.             regex = regex.replace("?", ".?");
25.             if (((String) arrayList.get(i)).matches(regex)) {
26.               result.add(true);
27.               break;
28.             } else {
29.               if (i == arrayList.size() - 1) {
30.                 result.add(false);
31.               }
32.             }
33.           }
34.         } else {
35.           if (i == arrayList.size() - 1) {
36.             result.add(false);
37.           }
38.         }
39.       }
40.     } else {
41.       result.add(false);
42.     }
43.   }
```

```
44.     }
45.  }
```

Listing 16-10: EqualOperator-class.

16.2.6 Boolean Operators

Two Boolean operators have been implemented, the AND- and the OR-operator. Both operators need two lists of Booleans to operate. This is illustrated by the constructor from the 'AndOperator'-class in listing 16-11.

```
1.  public AndOperator(List<Boolean> leftOperand, List<Boolean> rightOperand) {
2.      this.leftOperand = leftOperand;
3.      this.rightOperand = rightOperand;
4.  }
```

Listing 16-11: AndOperator-constructor.

The logical operator AND, also known as a logical conjunction, results in true if both of its operands are true, otherwise the result will be false. The logical operator OR, also known as a logical disjunction, results in true if at least one of its operands is true, otherwise the result will be false. The 'AndOperator'-class returns a list of Booleans. This is shown in listing 16-12. It loops through the list of left and right operands simultaneously. If both the left and right operand are true, the value 'true' will be added to the list of Booleans this method returns. If one of the operands is false, the value 'false' will be added to that list.

```
1.  public List<Boolean> getResult() {
2.  List<Boolean> result = new ArrayList<Boolean>();
3.  for (int i = 0; i < leftOperand.size(); i++) {
4.      if (leftOperand.get(i) && rightOperand.get(i)) {
5.          result.add(true);
6.      } else {
7.          result.add(false);
8.      }
9.  }
10. return result;
11. }
```

Listing 16-12: AndOperator-class.

16.2.7 'CullList'-class

The 'CullList'-class removes items from a list. Which items to remove is based on a list of Booleans. Therefore this class needs two lists to operate, a list of objects and a list of Booleans. This is illustrated by the constructor, which is shown in listing 16-13.

```
1.  public CullList(List<Object> objectList, List<Boolean> booleanList) {
2.      this.objectList = objectList;
3.      this.booleanList = booleanList;
4.  }
```

Listing 16-13: CullList-operator.

The list of Booleans will probably be generated by the relational or the Boolean operators. The 'CullList'-class loops through the list of objects and the list of Booleans simultaneously. If the Boolean value is equal to 'true' the related object will be added to a new list. However if the Boolean value equals 'false', then the related object will be ignored and the next Boolean value will be evaluated immediately. This procedure is executed by the code shown in listing 16-14.

```
1.  public List<Object> getResult() {
2.  List<Object> result = new ArrayList<Object>();
3.  for (int i = 0; i < objectList.size(); i++) {
4.      if (booleanList.get(i)) {
5.          result.add(objectList.get(i));
6.      }
7.  }
8.  return result;
9.  }
```

Listing 16-14: CullList-class.

16.2.8 'FlattenList'-Class

The 'FlattenList'-class converts a two dimensional list into a one dimensional list. This operation needs to be executed before a list of results is returned to the end-user. When the end-user for example wants to retrieve all GlobalIds of all doors, those GlobalIds will be stored in a two dimensional list. This class converts that list of lists of GlobalIds into one list of GlobalIds. The constructor of this class is in listing 16-15.

```
1.  public FlattenList(List<ArrayList> arrayListList) {
2.  this.arrayListList = arrayListList;
3.  }
```

Listing 16-15: FlattenList-operator.

The ‘FlattenList’-class loops through the list of lists. If the second list contains any objects, those objects will be added to a new list. However if the second list contains no items, the next list in the first list will be examined. The code for this procedure is shown in listing 16-16.

```
1.  public List<Object> getResult() {
2.  List<Object> objectList = new ArrayList<Object>();
3.  for (ArrayList arrayList : arrayListList) {
4.      if (arrayList.size() > 0) {
5.          for (Object object : arrayList) {
6.              objectList.add(object);
7.          }
8.      }
9.  }
10. return objectList;
11. }
```

Listing 16-16: FlattenList-class.

16.2.9 ‘GetRelatedObjectsOperator’-Class

This class loops through a list of objects and collects all objects directly related to those objects. These objects are stored in a list and depending on what the value of the integer it was passed it repeats this process. It loops through the objects just collected and retrieves all directly related objects. When such an object has not yet been stored to the list, it is stored to the list, otherwise it is ignored.

16.3 Integration of Transition Classes into the ANTLR Grammar

The classes described above provide all the functionality to perform queries on a building information model. By using them it is possible to extract the values of attributes and properties, to compare those values to a character string or number and to create a list of objects which meet certain criteria's.

These classes were developed to form a bridge between the BimQL grammar and the BimServer.org functionality. The classes above rely often on the classes and methods only present within the BimServer.org-platform. For example the generation algorithms for the 'GetAttribute'-classes and the actual 'GetAttribute'-classes itself cannot exist independent of the BimServer.org-platform. This relation has been discussed above. In this part the relation between the transition classes and the ANTLR-grammar is discussed. This part explains how the grammar creates instances of the transition classes and how the grammar get the query results by making use of those instances. For each grammar rule it is explained which transition classes are instantiated and how that is done.

16.3.1 'Bimql'-rule

The 'bimql'-rule (Listing 16-17) is the start of every query. It is passed an IFC model and it returns a list of objects. That list of objects will be the result of the query. Lines 3 through 7 retrieve all the objects from the model which type equals IfcRoot or which type is a subtype of IfcRoot. These objects are stored in a new list and passed on to the 'select'-rule.

```
1.  bimql [IfcModel ifcModel] returns [List<Object> bimqlReturns] :
2.  {
3.      List<IfcRoot> ifcRootList = new ArrayList<IfcRoot>();
4.      ifcRootList = ifcModel.getAllWithSubTypes(IfcRoot.class);
5.      List<Object> objectList = new ArrayList<Object>();
6.      for (IfcRoot ifcRoot : ifcRootList) {
7.          objectList.add(ifcRoot);
8.      }
9.  }
10. select [objectList] {
11.     $bimqlReturns = $select.selectReturns;
12. }
13. ;
```

Listing 16-17: Bimql-rule.

16.3.2 'Select'-Rule

The 'select'-rule (Listing 16-18) is passed a list of objects by the 'bimql'-rule and the result of this rule will again be returned to the 'bimql'-rule. The 'select'-rule lets the user choose a variable. The list of objects it was passed will now be associated with this variable.

```
1.  select [List<Object> objectList] returns [List <Object> selectReturns] :
2.  'Select' VARIABLE {
3.      $selectReturns = $select.objectList;
4.      hashMapObjectList.put($VARIABLE.text, objectList);
5.  }
6.  (where[$VARIABLE.text] {
7.      $selectReturns = hashMapObjectList.get($VARIABLE.text);
8.  })?
9.  (cascade {
10.     $selectReturns = hashMapObjectList.get($cascade.cascadeReturns);
11. })*
12. set?
13. ;
```

Listing 16-18: Select-rule.

16.3.3 'Cascade'-Rule

The 'cascade'-rule (Listing 16-19) can instantiate the 'GetEntityTypeMain'-class, the 'GetAttributeMain'-class, the 'GetPropertyMain'-class and the 'GetRelatedObjectsOperator'-class. In this rule these four classes make a new selection which is based on an already existing selection. The already existing selection could for example be all the windows of a certain height. The new selection would consist of the openings in which these windows are located.

The 'Flatten'-class can be instantiated multiple times. The new selection, the selection which contains the openings, is flattened in these lines. The selection which originally was a two dimensional list will just be a one dimensional list after these lines have been executed.

The option to omit the 'Attribute'- and 'Property'-tokens has been disabled, while this option is not yet operating as it should operate. Some problems exist distinguishing the 'String'-token and for example the 'Attribute'-token. This problem will be solved soon.

```
1.  cascade returns [String cascadeReturns] :
2.      'Select' VARIABLE1 = VARIABLE ':=' VARIABLE2 = VARIABLE (
3.          {
4.              List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE2.text));
5.              hashMapObjectList.put($VARIABLE1.text, objectList);
6.              $cascadeReturns = $VARIABLE1.text;
7.          }
8.          |'.EntityType' {
```

```
9.              List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE2.text));
10.             GetEntityTypeMain getEntityTypeMain = new GetEntityTypeMain(objectList);
11.             List<ArrayList> arrayListList = new
    ArrayList<ArrayList>(getEntityTypeMain.getResult());
12.             objectList.clear();
13.             FlattenList flattenList = new FlattenList(arrayListList);
14.             objectList = flattenList.getResult();
15.             hashMapObjectList.put($VARIABLE1.text, objectList);
16.             $cascadeReturns = $VARIABLE1.text;
17.         }
18.         |'.Attribute.' string1 = STRING {
19.             List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE2.text));
20.             GetAttributeMain getAttributeMain = new GetAttributeMain(objectList,
    $string1.text);
21.             List<ArrayList> arrayListList = new
    ArrayList<ArrayList>(getAttributeMain.getResult());
22.             objectList.clear();
23.             FlattenList flattenList = new FlattenList(arrayListList);
24.             objectList = flattenList.getResult();
25.             hashMapObjectList.put($VARIABLE1.text, objectList);
26.             $cascadeReturns = $VARIABLE1.text;
27.         }
28.         |'.Property.' string2 = STRING {
29.             List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE2.text));
30.             GetPropertyMain getPropertyMain = new GetPropertyMain(objectList,
    $string2.text);
31.             List<ArrayList> arrayListList = new
    ArrayList<ArrayList>(getPropertyMain.getResult());
32.             objectList.clear();
33.             FlattenList flattenList = new FlattenList(arrayListList);
34.             objectList = flattenList.getResult();
35.             hashMapObjectList.put($VARIABLE1.text, objectList);
36.             $cascadeReturns = $VARIABLE1.text;
37.         }
38.         | '.' string3 = STRING {
39.             System.out.println("Sorry, not yet implemented!!!");
40. //          List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE2.text));
41. //          GetAttributeMain getAttributeMain = new GetAttributeMain(objectList,
    $string3.text);
42. //          List<ArrayList> attributeArrayListList = new
    ArrayList<ArrayList>(getAttributeMain.getResult());
43. //          GetPropertyMain getPropertyMain = new GetPropertyMain(objectList,
    $string3.text);
44. //          List<ArrayList> propertyArrayListList = new
    ArrayList<ArrayList>(getPropertyMain.getResult());
45. //          objectList.clear();
46. //          FlattenList attributeFlattenList = new FlattenList(attributeArrayListList);

47. //          objectList = attributeFlattenList.getResult();
48. //          FlattenList propertyFlattenList = new FlattenList(propertyArrayListList);

49. //          objectList.addAll(propertyFlattenList.getResult());
50. //          hashMapObjectList.put($VARIABLE1.text, objectList);
51. //          $cascadeReturns = $VARIABLE1.text;
52.         }
53.         |(('.*')|('.*(' INTEGER? TRUE?')')) {
```

```
54.             List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE2.text));
55.             GetRelatedObjectsOperator getRelatedObjectsOperator = new
    GetRelatedObjectsOperator(objectList,
    Integer.parseInt($INTEGER!=null?$INTEGER.text:"1"), $TRUE!=null?true:false);

56.             List<ArrayList> getRelatedObjectsOperatorArrayListList = new
    ArrayList<ArrayList>(getRelatedObjectsOperator.getResult());
57.             objectList.clear();
58.             FlattenList flattenList = new
    FlattenList(getRelatedObjectsOperatorArrayListList);
59.             objectList = flattenList.getResult();
60.             hashMapObjectList.put($VARIABLE1.text, objectList);
61.             $cascadeReturns = $VARIABLE1.text;
62.         }
63.     )
64.     (where[$VARIABLE1.text]
65.     )?
```

Listing 16-19: Cascade-rule.

16.3.4 'Where'-Rule

The 'where'-rule (Listing 16-20) makes use of the 'CullList'-class. The list of objects which the 'where'-rule retrieves from the 'hashMapObjectList' is culled by an instance of this class. The list of Booleans needed for this operation is fetched from the 'statement'-rule.

```
1.  where [String string] :
2.  'Where' statement {
3.      CullList cullList = new CullList(hashMapObjectList.get($string),
    $statement.statementReturns);
4.      hashMapObjectList.put($string, cullList.getResult());
5.  }
6.  ;
```

Listing 16-20: Where-rule.

16.3.5 'Set'-Rule

The 'set'-rule (Listing 16-21) is the rule that instantiates the 'setAttributeMain'-class. This happens in line 3.

```
1.  set :
2.  'Set' VARIABLE1 = VARIABLE '.Attribute.' string1 = STRING ':=' (string2 = INTEGER |
    string2 = REAL | string2 = STRING) {
3.      SetAttributeMain setAttributeMain = new
    SetAttributeMain(hashMapObjectList.get($VARIABLE1.text), $string1.text, $string2.text);
4.      setAttributeMain.setAttribute();
5.  }
6.  ;
```

Listing 16-21: Set-rule.

16.3.6 'Statement'-Rule

The 'statement'-rule (Listing 16-22) instantiates both the 'AndOperator'-class and the 'OrOperator'-class. This happens in line 8 and in line 14. Notice the '$statementReturns'-variable in line 4. If only one relation is present in the this part of the query, the result of that query is immediately returned by this rule. However when a second relation is present and therefore the 'AND'- or 'OR'-operator is used, this variable is not instantly returned, instead line 7 or line 13 transform it into a parameter which is passed into an instance of the 'AndOperator'-class or the 'OrOperator'-class.

```
1.  statement returns [List<Boolean> statementReturns] :
2.  relation1 = relation {
3.      List<Boolean> firstBooleanList = new ArrayList<Boolean>($relation1.relationReturns);
4.      $statementReturns = firstBooleanList;
5.  }
6.  ('And' relation2 = relation {
7.      List<Boolean> tempBooleanList = new ArrayList<Boolean>($statementReturns);
8.      AndOperator andOperator = new AndOperator(tempBooleanList,
    $relation2.relationReturns);
9.      List<Boolean> andBooleanList = new ArrayList<Boolean>(andOperator.getResult());
10.     $statementReturns = andBooleanList;
11. }
12. |'Or' relation3 = relation {
13.     List<Boolean> tempBooleanList = new ArrayList<Boolean>($statementReturns);
14.     OrOperator orOperator = new OrOperator(tempBooleanList,
    $relation3.relationReturns);
15.     List<Boolean> orBooleanList = new ArrayList<Boolean>(orOperator.getResult());
16.     $statementReturns = orBooleanList;
17. })*
18. ;
```

Listing 16-22: Statement-rule.

16.3.7 'Relation'-Rule

The 'relation'-rule (Listing 16-23) can call one of six classes which define the relational operators.

```
1.  relation returns [List<Boolean> relationReturns] :
2.  relationleft (
3.      '=' relationright1 = relationright {
4.          EqualOperator equalOperator = new EqualOperator($relationleft.relationleftReturns,
    $relationright1.relationrightReturns);
5.          List<Boolean> booleanList = new ArrayList<Boolean>(equalOperator.getResult());
6.          $relationReturns = booleanList;
7.      }
8.      | '/=' relationright2 = relationright {
9.          InEqualOperator inEqualOperator = new
    InEqualOperator($relationleft.relationleftReturns, $relationright2.relationrightReturns);
```

```
10.         List<Boolean> booleanList = new
    ArrayList<Boolean>(inEqualOperator.getResult());
11.         $relationReturns = booleanList;
12.     }
13.     | '<' relationright3 = relationright {
14.         LessOperator lessOperator = new LessOperator($relationleft.relationleftReturns,
    $relationright3.relationrightReturns);
15.         List<Boolean> booleanList = new ArrayList<Boolean>(lessOperator.getResult());
16.         $relationReturns = booleanList;
17.     }
18.     | '<=' relationright4 = relationright {
19.         LessEqualOperator lessEqualOperator = new
    LessEqualOperator($relationleft.relationleftReturns, $relationright4.relationrightReturns);
20.         List<Boolean> booleanList = new
    ArrayList<Boolean>(lessEqualOperator.getResult());
21.         $relationReturns = booleanList;
22.     }
23.     | '>=' relationright5 = relationright {
24.         GreaterEqualOperator greaterEqualOperator = new
    GreaterEqualOperator($relationleft.relationleftReturns,
    $relationright5.relationrightReturns);
25.         List<Boolean> booleanList = new
    ArrayList<Boolean>(greaterEqualOperator.getResult());
26.         $relationReturns = booleanList;
27.     }
28.     | '>' relationright6 = relationright {
29.         GreaterOperator greaterOperator = new
    GreaterOperator($relationleft.relationleftReturns, $relationright6.relationrightReturns);
30.         List<Boolean> booleanList = new
    ArrayList<Boolean>(greaterOperator.getResult());
31.         $relationReturns = booleanList;
32.     }
33. )
34. ;
```

Listing 16-23: Relation-rule.

16.3.8 'Relationleft'-Rule

Not only the 'cascade'-rule can instantiate the 'GetEntityTypeMain'-class, the 'GetAttributeMain'-class, the 'GetPropertyMain'-class and the 'GetRelatedObjectsOperator'-class. The 'relationleft'-rule (Listing 16-24) is also able to do that. The 'cascade'-rule applied these instantiations to make a new selection, this rule however performs the same action to acquire information about the current selection. This information is then evaluated by the 'relation'-rule.

```
1.  relationleft returns [List<ArrayList> relationleftReturns] :
2.  VARIABLE '.EntityType' {
3.      List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE.text));
4.      GetEntityTypeMain getEntityTypeMain = new GetEntityTypeMain(objectList);
```

```
5.      List<ArrayList> arrayListList = new
    ArrayList<ArrayList>(getEntityTypeMain.getResult());
6.      $relationleftReturns = arrayListList;
7.  }
8.  | VARIABLE '.Attribute.' STRING {
9.      List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE.text));
10.     GetAttributeMain getAttributeMain = new GetAttributeMain(objectList, $STRING.text);
11.     List<ArrayList> arrayListList = new
    ArrayList<ArrayList>(getAttributeMain.getResult());
12.     $relationleftReturns = arrayListList;
13. }
14. | VARIABLE '.Property.' STRING {
15.     List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE.text));
16.     GetPropertyMain getPropertyMain = new GetPropertyMain(objectList, $STRING.text);
17.     List<ArrayList> arrayListList = new
    ArrayList<ArrayList>(getPropertyMain.getResult());
18.     $relationleftReturns = arrayListList;
19. }
20. |   VARIABLE '.' STRING {
21.     System.out.println("Sorry, not yet implemented!!!");
22. //      List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE.text));//
23. //      GetAttributeMain getAttributeMain = new GetAttributeMain(objectList,
    $STRING.text);
24. //      List<ArrayList> arrayListListA = new
    ArrayList<ArrayList>(getAttributeMain.getResult());//
25. //      GetPropertyMain getPropertyMain = new GetPropertyMain(objectList,
    $STRING.text);
26. //      List<ArrayList> arrayListListP = new
    ArrayList<ArrayList>(getPropertyMain.getResult());//
27. //      List<ArrayList> arrayListList = new ArrayList<ArrayList>();
28. //      arrayListList.addAll(arrayListListA);
29. //      arrayListList.addAll(arrayListListP);
30. //      $relationleftReturns = arrayListList;
31. }
32. | VARIABLE ('.*.EntityType' | '.*(' INTEGER? TRUE?').EntityType') {
33.     List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE.text));
34.     GetRelatedObjectsOperator getRelatedObjectsOperator = new
    GetRelatedObjectsOperator(objectList,
    Integer.parseInt($INTEGER!=null?$INTEGER.text:"1"), $TRUE!=null?true:false);
35.     List<ArrayList> getRelatedObjectsOperatorArrayListList = new
    ArrayList<ArrayList>(getRelatedObjectsOperator.getResult());
36.     List<ArrayList> arrayListList = new ArrayList<ArrayList>();
37.     for (ArrayList arrayList : getRelatedObjectsOperatorArrayListList) {
38.         GetEntityTypeMain getEntityTypeMain = new GetEntityTypeMain(arrayList);
39.         List<ArrayList> GetEntityTypeMainArrayListList = new
    ArrayList<ArrayList>(getEntityTypeMain.getResult());
40.         FlattenList flattenList = new FlattenList(GetEntityTypeMainArrayListList);
41.         arrayListList.add(flattenList.getResult());
42.     }
43.     $relationleftReturns = arrayListList;
44. }
45. | VARIABLE ('.*.Attribute' | '.*(' INTEGER? TRUE?').Attribute.') STRING {
46.     List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE.text));
```

```
47.     GetRelatedObjectsOperator getRelatedObjectsOperator = new
    GetRelatedObjectsOperator(objectList,
    Integer.parseInt($INTEGER!=null?$INTEGER.text:"1"), $TRUE!=null?true:false);
48.     List<ArrayList> getRelatedObjectsOperatorArrayListList = new
    ArrayList<ArrayList>(getRelatedObjectsOperator.getResult());
49.     List<ArrayList> arrayListList = new ArrayList<ArrayList>();
50.     for (ArrayList arrayList : getRelatedObjectsOperatorArrayListList) {
51.         GetAttributeMain getAttributeMain = new GetAttributeMain(arrayList,
    $STRING.text);
52.         List<ArrayList> getAttributeMainArrayListList = new
    ArrayList<ArrayList>(getAttributeMain.getResult());
53.         FlattenList flattenList = new FlattenList(getAttributeMainArrayListList);
54.         arrayListList.add(flattenList.getResult());
55.     }
56.     $relationleftReturns = arrayListList;
57. }
58. | VARIABLE ('.*.Property' | '.*(' INTEGER? TRUE?').Property.') STRING {
59.     List<Object> objectList = new
    ArrayList<Object>(hashMapObjectList.get($VARIABLE.text));
60.     GetRelatedObjectsOperator getRelatedObjectsOperator = new
    GetRelatedObjectsOperator(objectList,
    Integer.parseInt($INTEGER!=null?$INTEGER.text:"1"), $TRUE!=null?true:false);
61.     List<ArrayList> getRelatedObjectsOperatorArrayListList = new
    ArrayList<ArrayList>(getRelatedObjectsOperator.getResult());
62.     List<ArrayList> arrayListList = new ArrayList<ArrayList>();
63.     for (ArrayList arrayList : getRelatedObjectsOperatorArrayListList) {
64.         GetPropertyMain getPropertyMain = new GetPropertyMain(arrayList,
    $STRING.text);
65.         List<ArrayList> getPropertyMainArrayListList = new
    ArrayList<ArrayList>(getPropertyMain.getResult());
66.         FlattenList flattenList = new FlattenList(getPropertyMainArrayListList);
67.         arrayListList.add(flattenList.getResult());
68.     }
69.     $relationleftReturns = arrayListList;
70. }
```

Listing 16-24: Relatioleft-rule.

16.3.9 Remaining Rules

The other rules do not relate to a transition class. They do not instantiate such a class and therefore they are not mentioned separately.

```
1.  relationright returns [String relationrightReturns] :
2.  INTEGER {
3.      $relationrightReturns = $INTEGER.text;
4.  }
5.  |REAL {
6.      $relationrightReturns = $REAL.text;
7.  }
8.  |STRING {
9.      $relationrightReturns = $STRING.text;
10. }
11. ;
```

Listing 16-25: Relationright-rule.

```
1.  TRUE :
2.  '+'
3.  ;
4.
5.  FALSE :
6.  '-'
7.  ;
```

Listing 16-26: True and false-rules.

```
1.  VARIABLE :
2.  '$' STRING
3.  ;
```

Listing 16-27: Variable-rule.

```
1.  INTEGER :
2.  ('0'..'9')+
3.  ;
```

Listing 16-28: Integer-rule.

```
1.  REAL :
2.  INTEGER+ ('.' INTEGER+)?
3.  ;
```

Listing 16-29: Real-rule.

```
1.  STRING :
2.  (
3.      '\u002A'..'\u002A'
4.      |'\u003F'..'\u003F'
5.      |'\u0030'..'\u0039'
6.      |'\u0041'..'\u005A'
7.      |'\u005B'..'\u0060'
8.      |'\u0061'..'\u007A'
9.  )+
10. ;
```

Listing 16-30: String-rule.

```
1.  WS :
2.  (' ' | '\t' | '\n' | '\r' | '\f') + {$channel = HIDDEN;}
3.  ;
```

Listing 16-31: Whitespace-rule.

17 Appendix B: EG-ICE paper

The next few pages have been reserved for the paper, written together with Jakob Beetz. We presented this paper at the International Workshop for Intelligent Computing and Engineering, which was organized in Germany from the 4th till the 6th of July 2012.

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.

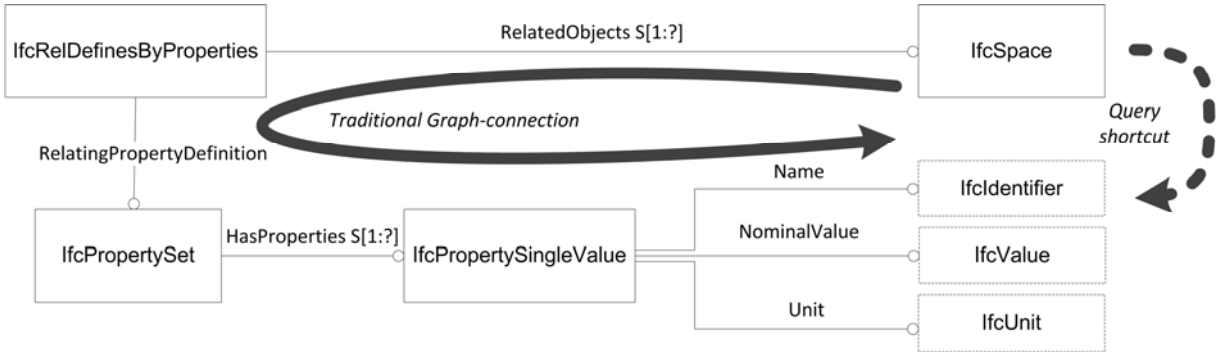


Figure 2: Query Shortcut against the Traditional Graph-connection

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"
```

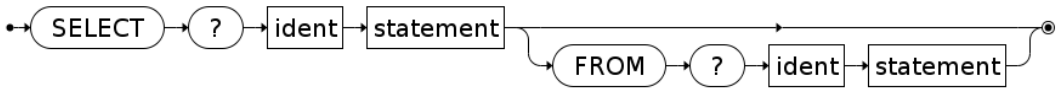


Figure 3: Select Syntax Diagram

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"
```

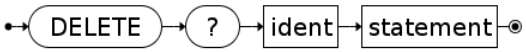


Figure 4: Delete Syntax Diagram

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"
```

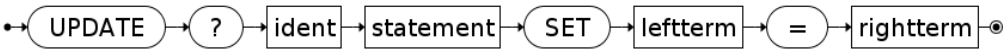


Figure 5: Update Syntax Diagram

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