Advanced Engineering Informatics 27 (2013) 444–456

Contents lists available at SciVerse ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

BIMQL – An open query language for building information models

Wiet Mazairac *, Jakob Beetz

Design Systems Group, Department of the Built Environment, Eindhoven University of Technology, PO Box 513, 600 MB Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 22 October 2012
Accepted 11 June 2013
Available online 25 July 2013

Keywords:
BIMQL
Domain Specific Language
Query Language
IFC
BIM
Building information model server

ABSTRACT

In this paper we present the on-going development of a framework for a domain specific, open query language for building information models. The proposed query language is intended for selecting, updating and deleting of data stored in Industry Foundation Classes 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 paper provides an overview of existing approaches, conceptual sketches of the language in development and documents the current state of implementation as a prototype plugin developed for the open source model server platform bimserver.org. We report on the execution of example test-cases to show the general feasibility of the approach chosen.

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. Over time, the exchange of information in the building process has become more complex. With increasing specialization, the number of stakeholders involved in a design and construction process has increased and so did the amount of information each actor generates. Current approaches to collaboration using digital information include shared building information models captured in vendor-neutral, standardized and interoperable model formats such as the Industry Foundation Classes (IFC) [1]. While most information exchange processes currently used in practice are still document-based [2–6], the use of model repositories that allow the integration of information via networks has recently seen a large increase in attention. These centralized or distributed repositories enable the structured and concurrent creation, integration and maintenance of large quantities of data from the various stakeholders involved in the planning and building process and the wide range of respective specialized software tools. These approaches have been advocated by the research community for many years [7–10] but have never really gained much practical relevance until recently. Even though no reliable survey data is available regarding their current practical application, the increasing number of commercial implementations of vendor specific or interoperable model repositories is a promising indicator for the increasing acceptance of these approaches gained among industry practitioners.

By merging models from different disciplines into common virtual models, the amount of information stored in such multi-domain repositories quickly becomes large and complex even for average projects. However, information needs of individual stakeholders in different process phases differ substantially. For example, a construction engineer does not require all the information available in the overall building model and is e.g. not interested in the detailed specifications of a suspended ceiling. This makes the extraction of partial model subsets or domain specific views on large models necessary.

A range of technologies with different conceptual approaches, intended use cases and target audiences have been devised to extract and manipulate partial model information. These approaches can be categorized as follows:

1. *Approaches for model instances* allow the individual inquiry and extraction of information on a per-model-instance level. Examples include the interactive selection of individual components or groups and classes of objects from a model. In AEC/FM, they often involve graphical user interfaces or are executed directly in the content creation applications such as CAD/BIM packages. They range from generic UI techniques like mouse picking, rubber-banding, selection sets and layer management to finely grained view definitions and object filtering that can be stored as templates for later reuse and thereby blur the border with schematic and conceptual approaches (category 2). These ad hoc approaches are often limited in flexibility, expressiveness and usefulness for automation in that they for example do not

* Corresponding author. Tel.: +31 681220502.
E-mail address: l.a.j.mazairac@tue.nl (W. Mazairac).

1474-0346/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.aei.2013.06.001

allow to select components based on individual class or attribute values (e.g. 'doors smaller than 1.01 m on the first floor', 'walls that are not made of brick') in straight forward ways. Often, they only reduce the quantity of instance individuals exposed to the user without altering the complexity of the model itself. In some cases though, they include pre-defined mappings and transformations into target formats such as reports or tabular data compilations.

2. *Schematic approaches* are used to transform model schemas themselves to support software engineering purposes by e.g. reducing model complexity or making implementations reusable in varying contexts. Two main directions can be identified in this category: Either they reduce complexity through extraction and/or transformation of schema subsets from larger schema models, e.g. through filtering on per-class and per-attribute levels or by flattening deep inheritance attribute-hierarchies. Or they take a reverse approach by composing granular conceptual building blocks often in a cascading fashion as is the case with XML namespaces or the 'USE [...] FROM' construct found in EXPRESS [11]. Schematic approaches are static and are used for repeated application on multiple models in re-occurring information takeoff scenarios, e.g. code-compliance checking, information hand-over and quantity takeoffs. They often include transformations and mappings of information into other, sometimes simplified and thus less complex schemas. However, up to now, the composition of such schematic views is a complex process which is not suitable for ad hoc per-project needs.
3. *Hybrid and querying approaches* operate on both instance and schematic levels at the same time. While model instance based approaches need existing schemas generated with the use of the schematic tools, hybrid approaches often have means to dynamically create new schemas or data structures, extract information from an instance model and transform the information into a new format. These approaches vary in their demands and applicability in daily AEC/FM practice: Hard-wired systems written in low-level programming languages or using high-level APIs such as the STEP SDAI [12] standard are impractical to address ad hoc needs by engineers and designers. Many query languages however lower the threshold to access information from model repositories significantly by abstracting and hiding some of the underlying data processing operations from the user. While some languages still require an intimate knowledge of low-level structures like model mappings into relational data base tables, others allow more focused, domain-specific queries on data. Their syntax and grammar is often oriented at natural language.

In Section 2 of this paper specific methods and related tools will be examined more in-depth and an overview is provided in Table 1.

For the IFC model format, the generation of such views has been addressed by a number of research initiatives and the standardization of such model subsets is addressed by the Model View Definition (MVD) effort [13]. MVDs are an essential part of the Information Delivery Manual (IDM) concept [14], in which a process-oriented, structured exchange of partial information is organized in automated ways [15]. However, these approaches rely on fixed sub sets of information which currently have to be defined, assembled and checked with specialized tools that are not easy to use by end-users. Currently, they do not allow an easy, project-specific assembly of ad hoc views. A review of MVD and its formal specifications can be found in Section 2.2. Although various approaches have been proposed for selecting and filtering data from a server on which a building information model is stored, an open, platform independent solution for creating such queries is currently not available.

A number of software environments designed to process building information models provide some way of selecting or filtering data. The bimserver.org project [16] for example enables end-users to extract parts of the model from the repository by selecting objects by their Globally Unique Identifiers (GUID), selecting all instances of a particular object class, using filters or by writing custom queries in Java that are compiled and executed on the server at runtime. Other tools such as the Solibri Model Viewer provide proprietary means to select, filter and check individual parts 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.

To address some of the shortcomings introduced in this section and elaborated in Section 2, we are introducing BIMQL (Building Information Model Query Language), a query language for IFC-based building information models to allow the selection and partial modification of model instances. Its aim is to provide a flexible means for ad hoc generation of partial data sets from IFC-based models based on practical information requirements. To ease the composition of queries the grammar and syntax of the language is designed to be adopted quickly by users already familiar with other generic languages. Domain-specific constructs add a layer of abstraction that hides some of the underlying complexities of the model. Although not fully implemented and tested yet, these abstraction mechanisms will be expanded in the future. In the long run, a possible combination with natural language constructs [17] will potentially lower the present requirements of in-depth knowledge of current and future IFC model schema structures.

Aimed at the composition of Service Oriented Architectures [18], several suggestions have been made to facilitate the remote access of model repositories via standardized network protocols [16,19–23]. However, most of these efforts are currently limited to administrative aspects such as access rights and version-management. They often only allow to access complete models, predefined views, or single objects at a time. For interoperability scenarios on the level of service-oriented tool-chains, BIMQL is suggested as a flexible, ad hoc way for information exchange similar to SPARQL [24] endpoints. BIMQL queries formulated spontaneously or composed from libraries can be used in combination with MVDs as a content selector in information request communicated via network protocols. The current implementation and integration into the bimserver.org framework allows remote BIMQL queries via SOAP [25], REST [26] and Protocol Buffers [27].

The paper is structured as follows: In the first section an introduction to the problem of information take off from complex models is provided and requirements for a query language are formulated. We then review related works and other approaches to the problem on both generic and domain-specific levels, identify open issues and position our work presented here in these contexts. The third section is dedicated to the design of the language itself as well as to the underlying mechanism of its prototypical implementation. Following this is an illustration of the specific features of BIMQL using a number of examples in Section 4. An indicative evaluation of the run-time performance of various models of different sizes is provided in the fifth section before concluding the paper with a critically discussion and evaluating of the proposed language and indicating future research and development directions.

1.1. BIM models captured in the IFC model format

The Industry Foundation Classes (IFC) is a vendor neutral and open model that defines more than 650 entities and a few

Table 1
Classification matrix of existing partial model generation tools.

Tool	Selection					Domain aspects			Technical aspects					
	By individual instance	By object class	By type (IfcMappedItem)	By value	By rule/constraint	Knowledge level required[a]	Domain specific	IFC support (out of the box)	Geometry topology	Recursion/arbitrary graphs	Bulk automation	Strongly typed	Manipulation [CRUD]	Schema transformation
CAD	x	x	–	–	–	1	x	O	x	–	–	n/a	X	–
Solibri	x	x	x	x	x	1/2	x	X	o	–	–	O	O	–
Navisworks	x	x	–	–	–	1/2	x	X	o	–	–	–	O	–
GTPPM	–	x	–	–	–	2	x	X	–	–	x	X	–	x
mvdXML	o	x	o	x	x	2	x	X	–	–	x	X	–	x
PQML	x	x	o	x	–	2	x	X	–	x	x	X	O	–
EQL	x	x	o	x	o	2	–	X	–	x	x	X	–	–
SQL	x	x	o	x	o	3	–	–	–	–	x	X	X	x
XSLT/XQuery	x	x	o	x	o	3	–	O	–	o	x	X	X	x
SPARQL	x	x	o	x	o	3	–	–	–	–	x	X	X	x
Gremlin	x	x	o	x	x	4	–	–	–	x	x	O	X	x
Linq	x	x	x	x	x	4	–	–	–	–	x	X	X	o
SDAI	x	x	x	x	x	4	–	X	–	x	x	X	X	o
Generic programming languages	x	x	x	x	x	4	–	O	o	x	x	O	X	o

x = supported; o = partially supported/by some implementations/through extensions/libraries; – = not supported.
[a] Knowledge level required (inclusive): 1 = basic understanding of CAD/BIM models, average domain expert; 2 = good to excellent understanding of the IFC model required; 3 = generic data modeling and querying knowledge required (DB design, XML, RDF etc.); 4 = programming knowledge required.

thousand attributes. It is based on the STEP EXPRESS [11] which is an ISO certified [28] schema modeling language used for large variety of domain models across different engineering industries. At its core, the notions of inheritance-supporting classes (called ENTITIYs in EXPRESS) and attributes of these classes that can have simple, derived or user-defined data types including references to other entity instances. To illustrate some of the EXPRESS modeling features as well as the specific design choices and modeling approaches used in the IFC model, information captured about doors is traced through the model as an example. The IfcDoor entity is a subtype of the IfcRoot entity, both shown in Listing 1: EXPRESS schema definition examples of the IfcDoor entity and its superclass IfcRoot.

A door in the IFC data model is thus not only specified by the width and height attributes related to the entity IfcDoor, but also by a GlobalId, OwnerHistory, Name and Description and other attributes inherited from the IfcRoot entity.

In addition to these explicit attributes defined on a schema level, the IFC model also allows the flexible definition of additional properties, grouped into property sets and assigned to the object (see Fig. 7) on the instance level. As shown in Listing 1 the height attribute is explicitly stated for the IfcDoor entity via an attribute, however to find other highly relevant information about particular objects like the fire rating, thermal transmittance etc. for doors another mechanism is present in the IFC model. Properties (as opposed to attributes which are defined on a schema level) can be defined and assigned to objects within an instance file. This is one of the generic extension and meta-modeling mechanism the IFC model provides to capture additional information not present in the common schema. However, in real-world implementations this is often used as weakly typed application-specific extension to e.g. facilitate round-tripping. The properties are grouped into sets (IfcPropertySet) and assigned to a particular object or type via the objectified relationship entity IfcRelDefinesByProperties, which requires a couple of hops during the traversal of the resulting model graph. The approach of objectified relationships is a modeling choice adopted throughout the IFC model: Instead of collecting object references through 'SET', 'LIST' and 'ARRAY' constructs on the attribute level, relationships such as decompositions, aggregation, connection etc. are separate entities that relate target objects amongst each other. This mechanism is also illustrated in Figure in which a wall, an opening and a window are connected by separate entities, both subtypes of the entity IfcRelConnect. A few thousand of these additional properties are standardized and governed by the buildingSMART organization and provided in the form of external XML files. This mechanism and the possibility to define arbitrary properties on-the-fly provide much needed flexibility but also impose a serious obstacle for schema-aware processing tools.

Because to date the IFC model is the most established neutral and open interoperability standard for building information modelling, it was chosen as the starting point for BIMQL. BIMQL streamlines the retrieval of objects from an IFC data model and provides shortcuts to elusive data that would otherwise require significant model navigation, for example retrieve properties of an object instance.

1.2. Requirements for a query language

Ideally, a framework developed to address the project-specific management of partial model information should enable the end user to interact with a building information model following the range of create, read, update and delete (CRUD) spectrum.

One of the specific aspects of building information captured in IFC models are its networked structures. Although many explicit relations can be accessed and manipulated in pre-defined,

```
ENTITY IfcDoor
SUPERTYPE OF (IfcDoorStandardCase)
SUBTYPE OF (IfcBuildingElement);
   OverallHeight: OPTIONAL IfcPositiveLengthMeasure;
   OverallWidth : OPTIONAL IfcPositiveLengthMeasure;
END_ENTITY;

ENTITY IfcRoot
ABSTRACT SUPERTYPE OF (ONEOF (IfcPropertyDefinition, IfcRelationship,
IfcObjectDefinition));
   GlobalId: IfcGloballyUniqueId;
   OwnerHistory: IfcOwnerHistory;
   Name: OPTIONAL IfcLabel;
   Description: OPTIONAL IfcText;
END_ENTITY;
```

Listing 1. EXPRESS schema definition examples of the IfcDoor entity and its superclass IfcRoot.

predictable tree-structures (e.g. the aggregation building elements through project-site-building-storey-building component) a number of constructs in these models exhibit the characteristics of bi-directional graph structures. Room-connectivity graphs describing the relation of individual spaces among each other, geometric transformation with arbitrarily deep nested local coordinate systems and the decomposition of building elements into sub-parts cannot be matched by pre-defined patterns that are common in traditional query languages. For such graph-like structures, recursive queries are necessary in order to find and manipulate information, enabling end users to collect information even from objects that are related to other objects via an arbitrary number of (objectified) relationships.

In order to make the language a useful tool for end user practitioners, a syntax close to the natural language should be aimed at. As an example, instead of using the lexically correct IFC class names such as IfcWallStandardCase (and discriminating or including, e.g. IfcWall from it) and IfcDoor, we would like to enable users to use natural language terms such as 'Wall', 'Walls' and 'Door'. Such a mechanism is prototypically demonstrated in the 'is-a' languages construct described in Section 3.6 which uses look-up dictionaries and structured vocabularies such as the International Framework for Dictionaries (IFD) [29,30], by which also a 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. An IfcWindow entitiy for example is not directly related to an IfcWall entity, instead three other entities are necessary to connect them (Fig. 1).

Often, even seemingly straight-forward relations between information entities in an IFC model require complex navigation of the underlying model that involves high-level technical knowledge of the underlying model schema in its respective implementation. One prominent example is the location of building elements on particular floors of a building, which can only be accessed by traveling nodes and edges (partially of inverse relationships) that should be hidden from the user. A prototypical implementation to address this issue with the BIMQL language is provided in Section 3.6. Apart from explicit 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 the external doors?" and "What is the shortest way to the exit?" without requiring intimate knowledge of a complex IFC schema are the targeted use cases of the proposed language.

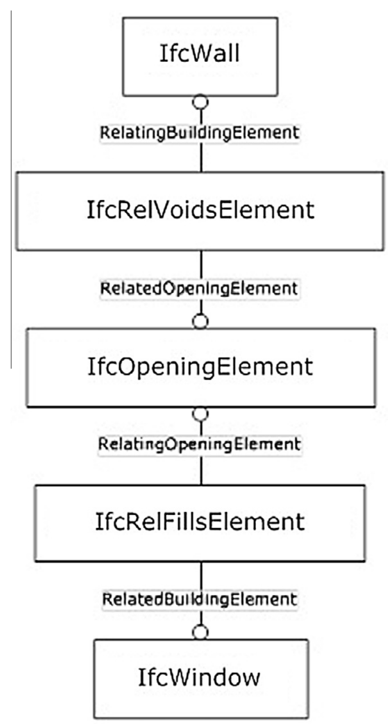


Fig. 1. Relationship between IfcWall and IfcWindow.

2. Related work

Over time a wide range of querying approaches have been developed that allow the extraction and manipulation of data in large models. They can be roughly divided into two groups. The first category includes generic querying approaches. These are more versatile, but are not able to address specific needs of interoperable building information modelling. Approaches specific to the domain of interoperability in the AEC/FM fall into the second category. In this section we provide an overview of these two categories of BIM querying approaches. Table 1 provides a classification matrix categorizing and comparing various features and aspects of the technologies aimed at partial model creation and interaction that are further elaborated upon in this section.

2.1. Generic querying approaches

Many of the existing database applications on the market use the Structured Query Language (SQL) as the standard language [31,32]. SQL was designed for managing data in a Relational Database Management System (RDBMS). SQL makes it possible to create, read, update and delete records, referred to as the CRUD principle [33]. Even though SQL in its various dialects is the most well-established query language to date, its applicability in an IFC context is limited. SQL is specified around set operations on data stored in tables. It would require traditional Object-Relational Mapping (ORM), of the more than 600 entities of the IFC model along with their thousands of attributes into individual tables. Earlier works have reported, that there are severe performance and scalability issues related to this approach [34,19,35]. Furthermore, the complexity of such queries would quickly increase beyond levels of feasibility. This is especially true for scenarios in which-non programming domain experts such as BIM managers are the end users. Even though the creation of nested, recursive queries using standard SQL is available through some implementations and extension modules, there is no widely accepted, standardized support that would be necessary to match the requirements formulated earlier.

Designed on top of the .NET platform, the Language Integrated Query (LINQ) [36,37] is Microsoft's technology to provide a language-level support mechanism for querying various types of data. These types include in-memory arrays and collections, and can also be applied to databases, XML documents and other forms of persisted data. Through its language-level integration, the versatility and expressiveness of the language is unmatched. Even though similar approaches have been proposed for other languages [38], no cross-platform mapping of a common standard is likely to be achieved. Furthermore, the generic nature of LINQ again limits the usefulness in an IFC context, since common, domain-specific queries such as provided earlier would require intimate knowledge of the model schema and result in queries written in a fully-fledged programming languages, not suitable for end user practitioners such as architects or structural engineers, who have no experience in programming.

The Resource Description Framework (RDF) is a directed, labeled graph data format for representing information on 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 have been proposed over the years [39,40], the SPARQL Protocol and RDF query language [24] is the most established one and has been officially standardized by the W3C. Especially relevant for the interoperability among distributed heterogeneous information systems is the standardization of service interfaces to SPARQL engines, referred to as SPARQL endpoints [41]. This building block of the Semantic Web effort enables the composition queries that span several repositories and thus effectively link information from different sources. In a BIM context, such a mechanism would allow the ad hoc connection of sub models that reside in repositories owned and operated by the various stakeholders in a building project. Such an approach could furthermore be used as a starting point for the inclusion of software services that inserts the results of specialized operations such as building regulation checking, performance calculations or quantity takeoffs. For the design of BIMQL, a number of features have been inspirational, including the syntactical means to specify and reference variables by the '?' token. Furthermore, extension mechanisms that have been proposed by the Jena ARQ implementation has been inspirational for the extension with custom functionality described in Sections 3.6 and 4.

The Object Constraint Language (OCL) [42] 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 et al. [43] designed as a generic query mechanism for STEP part 21 models. The language has a complex syntax and grammar which again renders its application by a large audience. Although proprietary implementation inspired by EQL are available in commercial applications its use is not widespread.

The Partial Model Query Language (PMQL) proposed by Adachi [44] aims to provide a general means for the selection, update, and deletion of partial model data that contains specific parts of product model data. Among the clear benefits the language design has over generic query languages described in Section 2.1, are its capabilities to construct recursive expressions. 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' operator, which has been proposed by Adachi to cope with recursive characteristics of IFC model graphs.

The Georgia Tech Process to product modeling [45] is a product modeling method to (semi-)automatically derive 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. In particular, the conceptual approach of hierarchical, modular Exchange Requirement (ER)/Functional Part (FP) trees differs substantially as does flattening supertype attributes into the exchange entity definitions in GTPPM. 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. [46] 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.

The technical standard to specify Model View Definitions (MVD) – mvdXML [47], which has been proposed as a standard by the buildingSMART organization and co-authored by the GMSD-author Weise, provides a rich set of filtering mechanisms to specify the sub-model information that should be contained in an MVD. Based on cascading concept templates that allow the definition of entity-, attribute- and property- selectors as well as constraints on values (''door height must be greater or equal to 1 m'') it covers a wide range of information exchange requirements. Even though authoring tools for the composition of such formal MVD specifications exist [48,49], which significantly accelerate the complex authoring of MVDs, the level of background knowledge required is very high. Both tools also need additional (not yet existing) processors to execute the actual model extraction based on mvdXML definitions. In the outlook Section 6.3 we show how these two technologies complement each other and could be combined to provide ad hoc querying capabilities on the one hand with standard-conform information compilation on the other hand.

Borrmann et al. [50] 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 Building Environment Rule and Analysis (BERA) language [51] is another powerful language that is tailored to the specific requirements of rule analysis that has been applied extensively in layout circulation checking scenarios of the court house use case also described in [7]. While its rule-checking focus differs from the query-focus of BIMQL domain specific features like the reference of attributes and properties through a [Entity].[attribute/property] pattern (e.g. "Space.GUID") bear some similarity in their usability and readability. However, contrary to BIMQL these attributes are hard-coded into the grammar of BERA, while a model-driven, introspective approach has been chosen in BIMQL.

The commercial application "Solibri Model Checker" [52] 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. Specification

The Backus-Naur Form [56] notation to describe the syntax and grammar of the domain specific query language BIMQL is provided in Listing 2. Note that the specification provided is currently limited to the 'select' and 'set' parts of the language features, whilst 'create' and 'delete' will be developed in the future. The implementation of the latter two language constructs requires a significant amount of additional research and software development work, because the consequences and side-effects of these operations are not trivial. 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 geometry, the openings covered by the particular window as well as the removal of indirect dependencies such as window profiles and material specifications in order to avoid unreferenced entities being carried on into the model evolution.

3.1. 'BIMQL'-rule

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. Currently the only top-level choice is 'select'. The 'set'-action is not a top-level choice, because it depends on the 'select'-action to determine which aspect of the model to manipulate. In the future other choices, for example 'create', can be added as subsequent instructions to the 'BIMQL'-rule.

3.2. 'Select'-rule

In the 'select'-rule (Fig. 2), the 'select'-token is followed by a variable designated by the question mark character (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 variable contained by the 'select'-rule can be followed by a 'where'-rule, multiple 'cascade'-rules or a 'set'-rule.

3.3. 'Where'-, 'cascade'- and 'set'-rule

The 'where'-rule (Fig. 3) can be used to define a conditional statement, referred to by the 'statement' keyword for the sake of brevity. Conditional statements make it possible to specify what to select and to narrow the selection down. Furthermore, the 'cascade'-instruction enables the user to make a new selection, based on an existing selection. The existing collection could for example be doors of a certain height. The cascade rule makes it possible to select the walls in which those doors are placed. The 'set'-rule is used when the value of an attribute needs to be changed.

3.4. 'Statement'-rule

Every 'where'-rule starts with a token that identifies it and is followed by a 'statement'-rule (Fig. 4). A statement can be composed of a single relation or can be broken down into a combination of several relations. If more relations are specified within one statement these relations are combined using the 'OR' and 'AND'-operations. These tokens indicate a disjunction or conjunction between the relations.

3.5. 'Relationleft'-rule

The 'relation'-rule is specified by a 'relationleft'- and a 'relationright'-rule and a collection of operator-tokens that separate them. The 'relationleft'-rule (Fig. 5) points to the entity type, property or attribute involved.

The 'relationleft'-rule starts with the variable defined earlier by the user. It is followed by the '.'-token. Similar to EXPRESS language rules, the '.'-token indicates a direct relation between its operands. This token is followed by an 'entitytype'-, an 'attribute'-, or a 'property'-token. The 'entitytype'-token can be used to specify the type of an entity (for example IfcDoor). The 'attribute'- and 'property'-token are both followed by the 'string'-rule. This string is used to specify the name of the attribute or property from which a value needs to be retrieved.

In contrast to an attribute which is directly defined and related to an entity on a fixed schema level, a property of an IFC model item is not directly connected to the entity it is related to. They are connected only indirectly via several edges and nodes in between them (see Fig. 7). Although an IfcPropertySingleValue can be retrieved by applying the 'attribute'-token multiple times and thus traversing the graph depicted in Fig. 7, this frequently used sequence of 'attribute'-tokens has been added as an explicit shortcut, the 'property'-operator (Listing 5).

Earlier versions of the implementation also ventured into leaving out the necessity to specify the type of relation (property or attribute). On the one hand, this further more decreases the required level of knowledge. On the other hand it comes with a price on the performance side since more branches of the graph will have to be traversed and processed. Future versions of the implementation might include setting parameters to allow the configuration of this trade-off by the user.

3.6. Plugin mechanism

Through a plugin mechanism, more domain-specific functionality can be added as explicit operators such as the geometric representation and other relations that take up multiple edges and nodes in the graph. These explicit domain specific query operators are not only syntactic sugar, limit the search and manual, explicit graph-traversal and -matching scope. In the case of searches this

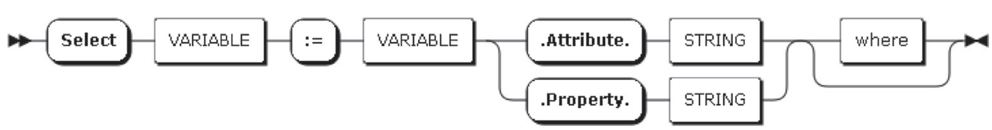
```
BIMQL         ::= select
select        ::= ‘Select’ VARIABLE where? cascade* set?
cascade       ::= ‘Select’ VARIABLE ‘:=‘ VARIABLE (‘.Attribute.’ STRING | ‘.Property.’
                     STRING) where?
where         ::= ‘Where’ statement
set           ::= ‘Set’ VARIABLE ‘.Attribute.’ STRING ‘:=‘ (INTEGER | REAL | STRING)
statement     ::= relation (‘And’ relation | ‘Or’ relation)*
relation      ::= relationleft (‘=‘ relationright | ‘/=‘ relationright | ‘<‘ relationright
                     |‘<=‘ relationright | ‘>=‘    relationright | ‘>‘ relationright)
relationleft  ::= (VARIABLE ‘.EntityType’ | VARIABLE ‘.Attribute.’ STRING | VARIABLE
                     ‘.Property.’ STRING | VARIABLE PLUGIN)
relationright ::= (INTEGER | REAL | STRING)
VARIABLE      ::= ‘$’ STRING
PLUGIN        ::= ‘:’ STRING
INTEGER       ::= ‘0..9’+
REAL          ::= INTEGER+ (‘.’ INTEGER+ )?
STRING        ::= (‘0..9’ | ‘A..Z’ | ‘a..z’ | ‘!’ | ‘#’ | ‘$’ | ‘%’ | ‘&’ | ‘^’ | ‘|’ |
                     ‘*’ | ‘+’ | ‘,’ | ‘-’ | ‘.’ | ‘/’ | ‘:’ | ‘;’ |‘<‘ | ‘=‘ | ‘>‘ | ‘?’
                     | ‘~’ | ‘`’ | ‘@’ | ‘_’)+
```

Listing 2. Backus-Naur form of the proposed BIMQL query language.

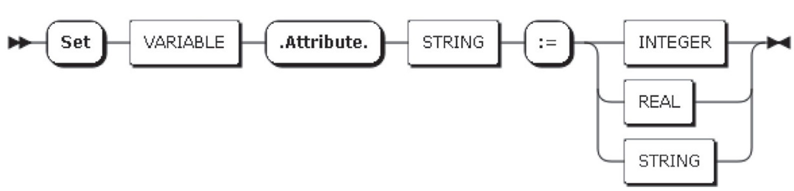

```
Select ?Everything
```

Fig. 2. Syntax diagram of the ‘select’ statement along with a BIMQL example selecting all entities of a model.


```
Select ?AllDoors
Where  ?AllDoors.EntityType = IfcDoor
```




```
Select  ?AllDoors
Where   ?AllDoors.EntityType = IfcDoor
Select  ?AllDoorsHeights := ?AllDoors.Attribute.OverallHeight
```

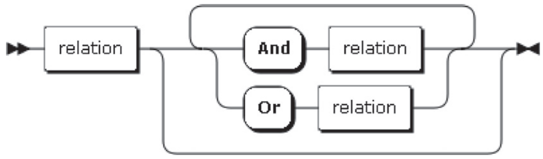



```
Select ?AllDoors
Where  ?AllDoors.EntityType = IfcDoor
Set    ?AllDoors.Attribute.Description := ProductXYZ
```

Fig. 3. Flow diagrams and illustrative examples of ‘where’, ‘select’ and ‘set’ rules.

also yields performance enhancements esp. on larger models. The plugin is triggered by the ‘:’ colon character, after which an arbitrary string is identifying the keyword associated with a specific plugin implementation.

As a proof of concept, two such plugins have been implemented: The “storey plugin” allows to filter building elements by building storey by going through the ‘IfcRelContainedInSpatialStructure’ objectified relationship instances of the model and looking whether the ‘RelatingStructure”s associated with an object (IfcProduct) name matches with the right-hand side. The “is-a plugin” (Listing 3) allows for a natural-language selection of building objects. It uses the ISO 12006-3:3-based buildingSMART Data Dictionary (bsDD) [53] to map natural language names to IFC entity definitions: The ‘IfcDoor’ class has been associated as one of the names of a concept (IfdSubject) which has the name “door” in the language “International English” as one of its names. The plugin implements a cached reverse-lookup that allows to retrieve names of the same concept in other languages (“Tür”, “Deur”, “Dør” etc.) and matches them with the right-hand side of the where statement. This allows for natural language selections of




```
Select ?AllDoors
Where  ?AllDoors.EntityType = IfcDoor
And    ?AllDoors.Attribute.Description = ProductXYZ
```

Fig. 4. Syntax diagram and illustrative example of the ‘relation’-rule and its combinations.

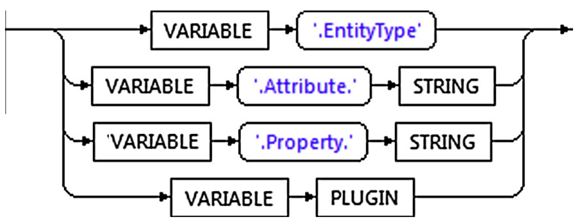


Fig. 5. ‘relationleft’-rule syntax diagram.

objects. The following examples selects all IfcDoors (searched for by its Dutch word “Deur”) and IfcWindows (searched for by its French term “fenêtre”) that are either on the first or second floor (searched for by the German names provided in the particular model.

Notice how the storey plugin can be triggerd by different keywords “Storey” (English) and “Verdieping” (Dutch), as an arbitrary number of keywords can be provided by a plugin registered into the query engine implementation at runtime. More details on the implementation of the extension mechanism can be found in Section 4.

3.7. ‘Relationright’-rule

The ‘relationright’-rule (Fig. 6) for the assignment of a comparison can be any string. If the string is numeric, it will be automatically compared with property definitions of simple and derived types provided in the property such as REAL, INTEGER or IfcPositiveLengthMeasure through automatic casting.

It is also possible to specify patterns by using asterisks, question marks and other terms familiar from regular expression terms [54]. The underlying functionality will try to match the pattern with the value the ‘relationleft’-rule returns. These patterns make is possible to e.g. return both ‘OverallHeight’ and ‘OverallWidth’ attributes of an IfcDoor by querying for ‘Overall∗’ or return the ‘SecurityRating’, ‘FireRating’ and ‘AccousticRating’ properties form the PSet_Door_common in one go by asking for ‘∗Rating’

3.8. Shortcuts

The introduction of shortcuts serve 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 (Listing 4) defined in the core schema constitutes a complex sub graph (Fig. 7). This requires several graph network ‘hops’ or nested iterations in procedural programming approaches and nested joins in traditional SQL based query languages.

The BIMQL-code in the above row of Listing 5 shows how BIMQL can be used to navigate the traditional graph-connection. Seven lines of code are needed. The first two lines select an object and the

```
Select ?Var1 Where ?Var1:Storey = Obergeschoss
    Or ?Var1:Verdieping = Erdgeschoss
   And ?Var1:is-a = Deur
    Or ?Var1:is-a = fenêtre
```

Listing 3. Natural language in a BIMQL query.

other five lines are required to retrieve the ‘volume’-property of that object. When the property shortcut is used (lower part) those five lines which were needed first are replaced by only one line.

In Section 1.1 the IFC model specification and specifically the concept of objectified relationships have been introduced. Objectified relationships could be a starting point for additional shortcuts. The next example, in which the boundary object for a given space are retrieved from the model (Listing 6) illustrates this principle. A space is related to its boundaries by the ‘IfcRelSpaceBoundary’-entity.

By introducing a new shortcut, based on the ‘IfcRelSpaceBoundary’-entity and named ‘SpaceBoundary’, the query not only becomes one line shorter, but also more comprehensible (Listing 7).

4. Implementation

The bimserver.org platform [16] already provides some means to extract partial building information models from a repository. These models can be downloaded after which they can be viewed or edited. An altered partial model can be uploaded to the server again on which it will be merged with the original model still present. Selections on individual model revisions can be made by specifying object IDs (including revision and authoring information), the IFC GUID, or all instances of a selected entity in the IFC schema. It is also possible to create custom queries by writing Java code which can be compiled and loaded during the runtime of the server, however the threshold to actually use this feature is high and the learning curve steep. In order to overcome this high entry threshold and because the bimserver.org is an accessible open source project we have chosen to integrate the proposed query language as a Domain Specific Language (DSL) that wraps the underlying querying mechanisms and hides the low-level technicalities from end-users. Next to the possibility to write Java code, the bimserver.org platform will be extended, so it will be possible to write BIMQL code.

The Model Driven Architecture (MDA) approach of software engineering is one of the architectural cornerstones of the bimserver.org framework. Instead of developing the source code itself, the programmer develops a model, which is used for automatically generating the source code. Although it increases the initial planning and writing resources required to produce the system that automatically generates source code from a model, this method increases portability, productivity and cross-platform interoperability. First the EXPRESS schema is converted to an Eclipse Modeling Framework (EMF) model. This model is then used to

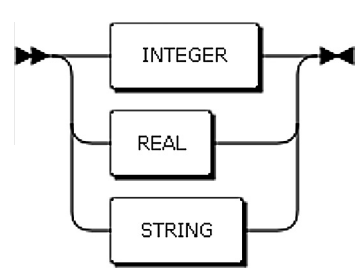


Fig. 6. ‘Relationright’-rule syntax diagram.

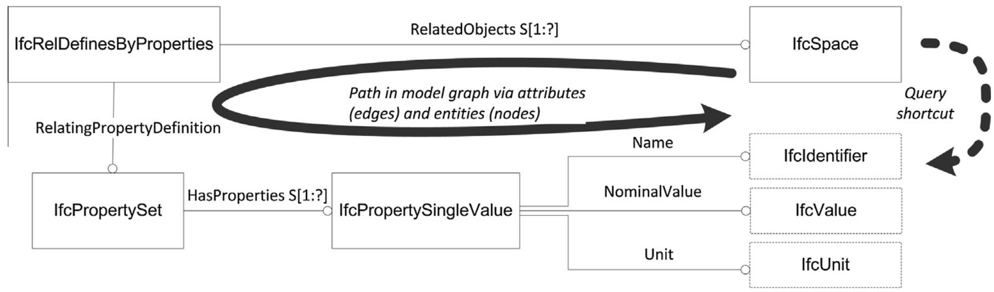


Fig. 7. Illustration of the mechanism to assign properties contained in a property set to an IfcObject (IfcSpace in this case) via the objectified relationship object IfcRelDefinesByProperties. The dashed line indicates a query shortcut.

```
ENTITY IfcSpace
SUBTYPE OF (IfcSpatialStructureElement);
       InteriorOrExteriorSpace      : IfcInternalOrExternalEnum;
       ElevationWithFlooring : OPTIONAL IfcLengthMeasure;
INVERSE
       HasCoverings: SET OF IfcRelCoversSpaces FOR RelatedSpace;
       BoundedBy   : SET OF IfcRelSpaceBoundary FOR RelatingSpace;
END_ENTITY;
```

Listing 4. Excerpt from the IFC2 × 3 model for spaces.

```
Select ?Var1
Where  ?Var1.Attribute.GlobalId = "87d87dffn47a90z"
Select ?Var2 := ?Var1.Attribute.IsDefinedBy
Select ?Var3 := ?Var2.Attribute.RelatingPropertyDefinition
Select ?Var4 := ?Var3.Attribute.HasProperties
Where  ?Var4.Attribute.Name = "Volume"
Select ?Var5 := ?Var4.Attribute.NominalValue
```

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

Listing 5. Comparison BIMQL statements selecting the ‘volume’-property with (upper part) and without (lower part) using the explicit ‘.property.’ shortcut.

```
Select ?Var1
Where  ?Var1.Attribute.GlobalId = "87d87dffn47a90z"
Select ?Var2 :=?Var1.Attribute.BoundedBy
Select ?Var3 :=?Var2.Attribute.RelatedBuildingElement
```

Listing 6. BIMQL query to retrieve the enclosures of a space without shortcut for objectified relationships.

generate Java classes for communicating with the bimserver.org database. This strategy was also applied in some of the implementation aspects of the Java classes underlying the BIMQL project. Here, bimserver.org Java classes which are based on the EMF Ecore model were used for generating some of the BIMQL Java classes. An overview of the system architecture of the bimserver.org framework and the BIMQL query plugin can be found in Fig. 8.

4.1. 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. In Listing 8 EXPRESS is used to describe the ‘IfcDoor’-entity.

```
Select ?Var1
Where  ?Var1.Attribute.GlobalId = "87d87dffn47a90z"
Select ?Var2 :=?Var1.SpaceBoundary
```

Listing 7. BIMQL query from Fig 14 with an additional shortcut for objectified relationships.

4.2. 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 9) 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.

4.3. 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. Listing 10 shows the bimserver.org Java class ‘IfcDoorImpl’ and one of its getters. The BIMQL Java classes are based on those methods.

4.4. BIMQL Java classes

The BIMQL Java classes are generated from the bimserver.org Java classes, are indirectly related to the IFC model specification and establish a link between the developed query language and the bimserver.org Java classes, which are talking to the database in which a building information model is stored. Listing 11 shows part of a BIMQL Java class. This specific class is based on the class in Listing 10.

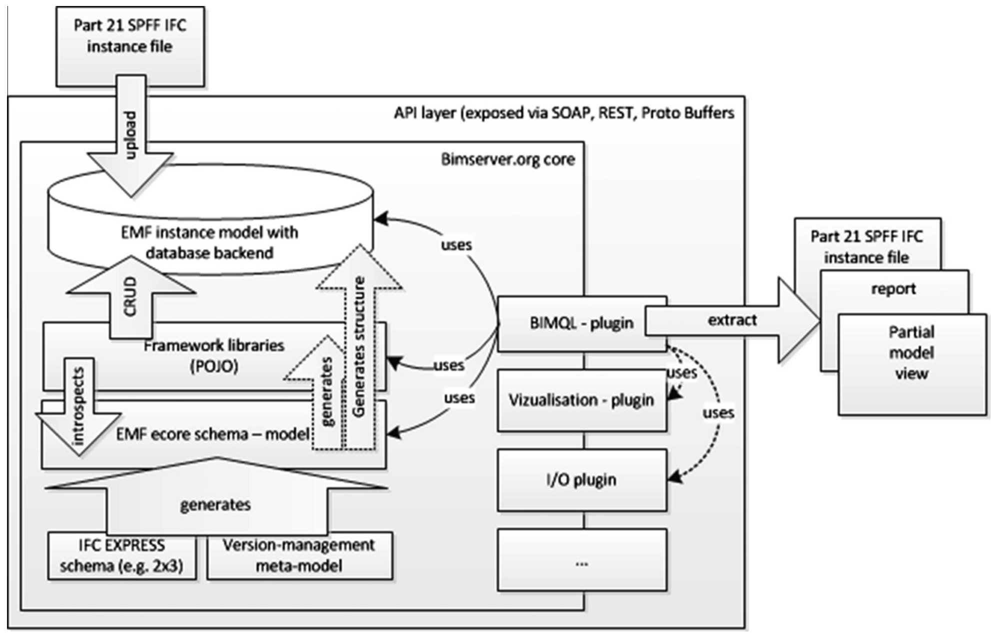


Fig. 8. System overview of the bimserver.org framework, the BIMQL plugin and the MDA layers involved.

```
ENTITY IfcDoor
    SUPERTYPE OF (IfcDoorStandardCase)
    SUBTYPE OF (IfcBuildingElement);
    OverallHeight: OPTIONAL IfcPositiveLengthMeasure;
    OverallWidth : OPTIONAL IfcPositiveLengthMeasure;
END_ENTITY;
```

Listing 8. EXPRESS schema definition of the IfcDoor entity.

4.5. Implementation of the grammar plugins

In order to make the BIMQL extendable a mechanism has been inserted that allows the execution of arbitrary grammar plugins through a plugin interface. The ':'-character serves as a 'magic token' in the 'plugin-rule' on the left-handside statement rule that marks an arbitrary variable keyword. Upon initialization of the BIMQL query language engine prototype, plugins can be loaded into the engine, that implement a´BIMQLGrammarPluginInterface' interface class. When the colon magic token is recognized by the parser (which is generated from a static ANTLR-grammar used in the prototype implementation) all registered plugins are queried for the variable keyword following the colon. If found, the particular evaluation method of the appropriate plugin is called. This way, not only arbitrary amounts of additional keywords can be plugged into the language making expansions through the community possible and shareable. Individual keywords can also be localized easily to add natural language integration. The ":onStorey"/":opVerdieping" example provided above is an example for this.

5. Examples

In addition to examples provided in Section 3, we will present a few examples in this section to demonstrate the use of this new language. 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.

5.1. Examples of selecting information

The first examples will retrieve parts of the IFC model. Parts of an IFC model can be all the windows, the first floor or a specific column, but a part can also be a list of numbers, for example all the doors and their dimensions. The example query in Listing 12 returns all spaces whose area is larger than 20 square meters. Notice the 'property'-operator. This operator provides a shortcut for retrieving the value of a property. 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.

The example in Listing 13 returns the floors areas of spaces whose GlobalIDs are known. The first two lines select the spaces of which the floor areas need to be retrieved. The last two lines perform a new query based on those first two lines. These two lines actually return the floor areas. This example returns a list of values, instead of only one value. BIMQL could be improved by adding explicit aggregation functions, which can for example locate the maximum or minimum value.

The final example in Listing 14 returns all doors which are too small to provide access to an operating room. The first two lines select all operating rooms. Lines 3 and 4 select all building elements which define the operating room space and the last line limits that selection to the doors with a certain dimension.

5.2. Examples of altering information via the 'Set' operation

The set feature makes it possible to change the value of an attribute. The example in Listing 15 changes the name of a space to 'Kitchen'.

Notice that currently no further consistency checks have been implemented to safeguard the integrity of the model when applying value modifications through the 'set' operation. Examples beyond those described in Section 5.2 include the modification of a value explicitly attributed to an entity that should be derived from its implicit geometric properties (OverallWidth of an IfcDoor) or the inadvertent deletion of references by list-redefinitions (IfcRepresentationItems in an IfcRepresenation).

```
<eClassifiers xsi:type="ecore:EClass" name="IfcDoor" eSuperTypes="#//IfcBuildingElement">
<eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallHeight"/>
<eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallHeightAsString"/>
<eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallWidth"/>
<eStructuralFeatures xsi:type="ecore:EAttribute" name="OverallWidthAsString"/>
</eClassifiers>
```

Listing 9. EMF model definition of the IfcDoor entity.

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

Listing 10. Part of the bimserver.org IfcDoor Java class.

```
public class SetAttributeSubIfcDoor {
    public void setAttribute() {
    ...
        else if (attributeName.equals("OverallHeight")) {
            ((IfcDoor) object).setOverallHeight(Double.parseDouble(attributeNewValue));
        }
    }
}
```

Listing 11. Part of the BIMQL IfcDoor Java class.

```
Select ?MySpace
Where  ?MySpace.EntityType = IfcSpace
And    ?MySpace.Property.NetFloorArea > 20
```

Listing 12. BIMQL example selects spaces based on the floor area.

```
Select ?mySpace
Where  ?mySpace.Attribute.GlobalID = 3Dn6BYWjfErxE1JocogMGQ Or
   ?mySpace.Attribute.GlobalID = 0tLttARJ1F1esyGJSeaTmd
Select ?netFloorArea
Where  ?netFloorArea := ?mySpace.property.NetFloorArea
```

Listing 13. BIMQL example returns the floor area of a space.

6. Summary and outlook

6.1. Summary

In this paper we introduced BIMQL, an extendable, open, domain specific query language for building information models. It can be used to select and update partial aspects in building information models. We described our conceptual approaches, the formal specification of the language, an implementation of the design as well as preliminary performance tests. The language has been designed and implemented on top of the bimserver.org platform even though it is generic enough to be adapted in other implementations of IFC-based modeling and development tools. At its current state of implementation, it allows the selection of objects and attributes based on their schema names or arbitrary properties stemming from standardized or custom property sets. While it is in principle possible to select objects connected to each other via arbitrary relationship edges in a STEP part 21 graph network, BIMQL introduces extendable, domain specific shortcuts to make these selection processes easier. Two shortcuts have been demonstrated that allow querying for properties related to objects and the containment of elements on building storeys through a single term instead of a number of sequential graph traversal statements (".property." and ":onStorey"). The flexible syntax plugin mechanism for the future extension of the BIMQL language through further operators by the community has been demonstrated by an example: The term ":is-a", allows the lookup of IFC schema names of objects (IfcDoor, IfcWindow, etc.) by names in other languages associated to the term (Tür, fenêtre, etc.) in the buildingSMART data dictionary.

Even though the language in its current state of specification and implementation is useful for a number of ad hoc and repeated sub-model extraction scenarios described in this paper, a number of limitations and desirable additions are going to be addressed by future research and development:

6.2. Advanced natural language capabilities

The search for model parts by their natural language terms is currently limited to singulars signifiers explicitly mapped in the buildingSMART data dictionary. More sophisticated natural language mappings could be achieved in future through a number of different approaches. These include the incorporation of more sophisticated structured vocabularies such as WordNet and DBPedia and their counterpart in other languages through Web services or derived cached mappings. Such mappings would also require more sophisticated mapping techniques.

6.3. Query patterns and combination with Model View Definition Templates

Another limitation of the current state of BIMQL is the necessity to request all required information aspects manually by defining individual query variables per desired aspect ($wall and $openingCoverInWall) or to gather this information by sequential queries or other form of post-processing. In future, rather than

```
Select ?OperatingRoom
Where  ?OperatingRoom.EntityType = IfcSpace And ?OperatingRoom.Attribute.Name = OR*
Select ?OperatingRoomSpaceBoundary := ?OperatingRoom.Attribute.BoundedBy
Select ?OperatingRoomSmallDoor :=
  ?OperatingRoomSpaceBoundary.Attribute.RelatedBuildingElement
Where  ?OperatingRoomSmallDoor.EntityType = IfcDoor
And    ?OperatingRoomSmallDoor.Attribute.OverallWidth < 150
```

Listing 14. More complex BIMQL example selecting all doors which are too small.

```
Select $MySpace
Where  $MySpace.GlobalID = 3Dn6BYWjfErxE1JocogMGQ
Set    ?MySpace.Name := Kitchen
```

Listing 15. BIMQL example changing the name of an object.

returning collections of objects or attribute values, it would be preferable to provide mechanisms to define and reuse aggregated information compilations on higher conceptual levels (e.g. get all geometrical representations of a building element or all information including postal address, mail, role etc. when querying for an IfcPerson). In addition to the possibility of building up repositories of frequently used query patterns and "stored procedures" one of the most interesting directions is the inclusion of Model View Definitions. More specific, the expected rapid growth of mvdXML definitions and their conceptual building blocks (templates) could be used to compile result sets to queries. Preliminary selection mechanisms and algorithms are present in the bimserver.org codebase and could be modified and adapted within a few person weeks.

6.4. Spatial queries

Similar to the existing plugin mechanism used for statements on the lefthand-side of comparison operators ("?var:is-a = Door"), such a mechanism could be added for other kinds of operators. Spatial query algebras and algorithms as outlined in [55] could implemented to allow searches for walls touching a particular other wall through specialized operators like ("?wall.attribute.GUID = [ID] And ?otherwalls:is-a = Wall And ?otherwalls:touches ?wall"). While the addition of the extension-mechanism for operators is straight-forward, the inclusion of the spatial reasoning capabilities themselves require much additional research and development work. It would then for example be possible to search for all south faced windows and all external doors on the ground floor (even if not explicitly stated, e.g. through a "is External" property) and determining the implicit 'ground floor' by relating it to the terrain elevation. While developing the spatial query functionality, simultaneously new shortcuts could be developed to accelerate the development of queries.

6.5. Distribution and service orientation

Rather than using centralized model repositories the approach of federated servers that contain domain specific sub-models (structural model, HVAC model) somewhere in a network structure has received much attention. Here, a special problem is to query several repositories at once, dealing with administrative issues (rights-management etc.) as well as the necessity to merge partial query results and the problem of inter-sub-model dependencies of information. It be interesting to extend BIMQL with capabilities either to explicitly address specific repositories for parts of the query. ("Get x from repository Alice, use x to retrieve y from Bob".)

Similarly, it would be interesting to be able to trigger remote services specializing in particular operations. ("Get surface area of x according to building regulation A from service Alice. Retrieve invalid objects from Bob"). In such a way BIMQL could even be elaborated as a content language for agent systems interacting in distributed networks.

6.6. Additional domain specific shortcuts

In addition to the shortcuts demonstrated here ("on storey" "property x in IFC property set") other frequently used paths and subgraphs could be abbreviated into shortcuts, e.g. simplifying the search for certain specific dependencies between entities. This would make it easier to find all doors or windows related to one wall for example. This is already possible now, however more than one line of query code is needed.

Even though the proposed language is far from feature complete, with BIMQL, we hope to provide a useful vantage point for future research, discussion and development on the development for end-user interfaces to complex building information models as well as the composition of service-oriented architectures.

Ackknowledgements

We would like to extend our 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.

References

[1] ISO/PAS 16739:2005 Industry Foundation Classes, Release 2x, Platform Specification (IFC2x Platform), 2005.
[2] B. Ingirige, G. Aouad, Awareness and Usage of Information Standards in the UK Construction Industry: A Survey by The SIENE Network, 2000.
[3] T. Froese, Z. Han, M. Alldritt, Study of information technology development for the Canadian construction industry, Canadian Journal of Civil Engineering 34 (2007) 817–829.
[4] N. Forcada, M. Casals, X. Roca, M. Gangolells, Adoption of web databases for document management in SMEs of the construction sector in Spain, Automation in Construction 16 (2007) 411–424.
[5] B.B.-G., S. Rice, An assessment of building information modeling value and use, in: CIB W078 2009, 2009.
[6] L. Khemlani, Top Criteria for BIM Solutions: AECbytes Survey Results, AECbytes, AECbytes Special Report, 2007.
[7] C. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors, John Wiley & Sons, 2011.
[8] T. Cerovsek, A review and outlook for a "Building Information Model" (BIM): a multi-standpoint framework for technological development, Advanced Engineering Informatics 25 (2011) 224–244.
[9] W. Shen, Q. Hao, H. Mak, J. Neelamkavil, H. Xie, J. Dickinson, et al., Systems integration and collaboration in architecture, engineering, construction, and facilities management: a review, Advanced Engineering Informatics 24 (2010) 196–207.
[10] N. Bakis, G. Aouad, M. Kagioglou, Towards distributed product data sharing environments – progress so far and future challenges, Automation in Construction 16 (2007) 586–595.
[11] D.A. Schenck, P.R. Wilson, Information Modeling: the EXPRESS Way, Oxford University Press, Inc., New York, NY, USA, 1994.

[12] ISO 10303-22:1998 Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 22: Implementation Methods: Standard Data Access Interface, 1998.
[13] M. Venugopal, C.M. Eastman, R. Sacks, J. Teizer, Semantics of model views for information exchanges using the industry foundation class schema, Advanced Engineering Informatics 26 (2012) 411–428.
[14] D. Davis, J. Karlsh/oj, R. See, An Integrated Process for Delivering IFC Based Data Exchange, BuildingSMART, 2012.
[15] J. Karlshoej, Process and building information modelling in the construction industry by using information delivery manuals and model view definitions, in: G. Gudnason, R.J. Scherer (Eds.), ECPPM 2012, Taylor & Francis Group, 2012.
[16] J. Beetz, L. van Berlo, R. de Laat, P. van den Helm, Bimserver.org – an open source IFC model server, in: Proceedings of the CIB W78 2010: 27th International Conference, Cairo, Egypt, 2010.
[17] R. Niemeijer, B. de Vries, J. Beetz, Constraint soup, in: Natural Language Processing and Knowledge Engineering (NLP-KE), 2010 International Conference On, 2010, pp. 1-6.
[18] T. Erl, Service-Oriented Architecture (SOA): Concepts, Technology, and Design, Prentice Hall, 2005.
[19] A. Kiviniemi, M. Fischer, V. Bazjanac, Integration of multiple product models: Ifc model servers as a potential solution, in: Proc. of the 22nd CIB-W78 Conference on Information Technology in, Construction, 2005.
[20] Open BIM Collaboration Format Web-Service API Proposal, n.d.
[21] B. Kumar, J.C. Cheng, L. McGibbney, Cloud computing and its implications for construction IT, in: Int. Computing Conf. on Civil and Building, Engineering, 2010.
[22] A. Redmond, A. Hore, M. Alshawi, R. West, Exploring how information exchanges can be enhanced through Cloud BIM, Automation in Construction 24 (2012) 175–183.
[23] Constructivity Server IFC Interface. <http://www.constructivity.com/cmserver.htm> (accessed 1.1.13).
[24] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF – Working Draft, 2005.
[25] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana, Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI, IEEE Internet Computing 6 (2002) 86–93.
[26] R.T. Fielding, R.N. Taylor, Principled design of the modern web architecture, ACM Transactions on Internet Technology (TOIT) 2 (2002) 115–150.
[27] Google Protocol Buffers Documentation, (n.d.).
[28] ISO10303-11:1994, Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 11: Description Methods: The EXPRESS Language Reference Manual, 1994.
[29] ISO12006-3, Building Construction – Organization of Information about Construction Works – Part 3: Framework for Object-Oriented Information, 2006.
[30] C. Lima, A. Zarli, G. Storer, Controlled vocabularies in the European construction sector: evolution, current developments, and future trends, in: Complex Systems, Concurrent Engineering, 2007, pp. 565–574.
[31] E.F. Codd, A relational model of data for large shared data banks, Communications of the ACM 13 (1970) 377–387.
[32] ISO/IEC 9075-1:2011 Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), (n.d.).
[33] J. Martin, Managing the Data-Base Environment, Prentice-Hall, 1983.
[34] 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.
[35] Y. Adachi, Overview of IFC Model Server Framework, EWork and EBusiness in Architecture, Engineering and Construction, 2002, p. 367.
[36] E. Meijer, B. Beckman, G. Bierman, LINQ: reconciling object, relations and XML in the .NET framework, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2006. pp. 706–706.
[37] P. Pialorsi, M. Russo, Introducing Microsoft's Linq, First Microsoft Press, Redmond, WA, USA, 2007.
[38] E. Wcisło, P. Habela, K. Subieta, A java-integrated object oriented query language, in: A. Abd Manaf, A. Zeki, M. Zamani, S. Chuprat, E. El-Qawasmeh (Eds.), Informatics Engineering and Information Science, Springer, Berlin Heidelberg, 2011, pp. 589–603.
[39] P. Haase, J. Broekstra, A. Eberhart, R. Volz, A Comparison of RDF query languages, in: Proc. Third International Semantic Web Conference, Springer, 2004, pp. 502–517.
[40] J. Bailey, F. Bry, T. Furche, S. Schaffert, Web and semantic web query languages: a survey, in: N. Eisinger, J. Maluszynski (Eds.), Reasoning Web, Springer, Berlin/Heidelberg, 2005. 95–95.
[41] K.G. Clark, L. Feigenbaum, E. Torres (Eds.), SPARQL Protocol for RDF, (n.d.).
[42] J.B. Warmer, A.G. Kleppe, The Object Constraint Language: Getting Your Models Ready for MDA, Addison-Wesley Professional, 2003.
[43] D. Koonce, L. Huang, R. Judd, EQL an express query language, Computers & Industrial Engineering 35 (1998) 271–274.
[44] Y. Adachi, Overview of partial model query language, in: Proc. of the 10th Int. Conf. on, Concurrent Engineering, 2003, pp. 549–555.
[45] G. Lee, R. Sacks, Generating IFC VIEWs and Conformance Classes using GTPPM, in: Proc. 11th International Conference on Computing in Civil and Building Engineering ICCCBE-XI, 2006, pp. 1715–1724.
[46] M. Weise, P. Katranuschkov, R.J. Scherer, Generalised model subset definition schema, in: Construction IT: Bridging the Distance, Proceedings of the CIB-W78 Workshop, 2003, p. 440.
[47] T. Chipman, T. Liebich, M. Weise, mvdXML – Specification of a Standardized Format to Define and Exchange Model View Definitions with Exchange Requirements and Validation Rules, buildingSMART, 2012.
[48] IfcDoc Tool – The New MVD Development Tool. <http://www.buildingsmart-tech.org/blogs/msg-blog/ifcdoc-the-new-mvd-development-tool> (accessed 2.15.13).
[49] ViewEdit Tool – Private Email Conversation with Matthias Weise, 2012.
[50] A. Borrmann, C. van Treeck, E. Rank, Towards a 3D Spatial Query Language for Building Information Models, in: H. Rivard, M.M. Cheung, H.G. Melhem, E.T. Miresco, R. Amor, F.L. Ribeiro (Eds.), 11th International Conference on Computing in Civil and Building Engineering ICCCBE-XI, Montreal, Canada, 2006, pp. 1374–1385.
[51] J. Lee, Building Environment Rule and Analysis (BERA) Language, PhD Thesis, Georgia Institute of Technology, 2011.
[52] Solibri Inc, Solibri Model Checker. <http://www.solibri.com/solibri-model-checker.html> (accessed 2.15.13).
[53] R. Grant, H. Bell, L. Bjorkhaug, A. Bjaaland, IFD Library White Paper, 2008.
[54] Mastering Regular Expressions, O'Reilly Media, Inc., 2006.
[55] A. Borrmann, J. Beetz, Towards spatial reasoning on building information models, in: Proc. of the 8th European Conference on Product and Process Modeling (ECPPM), 2010.
[56] ISO/IEC 14977:1996(E) Information technology – Syntactic Metalanguage – Extended BNF.