A Spatio-semantic Query Language for the Integrated Analysis of City Models and Building Information Models

S. Daum, A. Borrmann, T. H. Kolbe

Abstract Distinct semantic-geometric data models are applied in architecture/engineering/construction and geospatial domains. However, to make decisions within complex urban planning and engineering tasks these two domains and their data models must be combinded. Currently, the necessary joint information is created by converting data between the two domains. Because the employed modelling differs conceptually, there is a high risk of information loss durring these conversions. To overcome this issue, we present a spatio-semantic query language that allows analysis of IFC building information models and geospatial CityGML models in an integrated context. Rather than converting between IFC and CityGML, a holistic information space is realized by an intermediate layer that abstracts from the two schemas of spatio-semantic modelling.

1 Introduction

The examination of data from diverging scales is necessary for various planning tasks. The scales to be considered range from the detailed construction information of a single building up to the spatial information of an entire city or region. A typical urban planning task is the construction of a new subway line, which involves a large-scale alignment design of the track and detailed designs of new train

S. Daum
Chair of Computational Modeling and Simulation, Technische Universität München,
e-mail: simon.daum@tum.de

A. Borrmann
Chair of Computational Modeling and Simulation, Technische Universität München,
e-mail: andre.borrmann@tum.de

T. H. Kolbe
Chair of Geoinformatics, Technische Universität München,
e-mail: thomas.kolbe@tum.de

stations, escape shafts and traffic nodes (Borrmann et al., 2015). Currently, experts use different information sources for these scales, in particular building information modelling (BIM) systems and geographic information systems (GIS). For both domains, standardised data schemas for semantic and geometrical modelling exist. However, an integrated analysis of BIM and GIS data with respect to spatial and semantic criteria has not been possible to date.

More and more city models are being represented according to CityGML (Gröger and Plümer, 2014). This well-known data model is employed to represent partial and whole cities at five levels of detail (LOD) (Kolbe, 2009). In the most detailed level, LOD4, the interior of buildings and other types of construction are also represented. CityGML is designed to represent an as-built situation as captured by a survey. For this reason, the modelling principle of CityGML is based on representing the viewable surfaces of a building, e.g. expressed by classes such as *WallSurface* and *RoofSurface*.

In the architecture, engineering and construction (AEC) domain, the standardised data model industry foundation classes (IFC) is employed to realize open data exchange between BIM tools. In contrast to CityGML, the IFC schema focuses on the modelling of building elements and their relationships. The combination of semantic objects, explicitly stored relationships and geometry representations based on solid modelling meets the requirements of planning, constructing and operating buildings. Thus, an IFC model can be used as a single holistic data pool supporting all phases of a building's life cycle. Conceptually, the IFC model provides the foundation for BIM which promotes a new way of working in the construction domain, that is fully based on high-level digital models. Currently, BIM is gaining wide acceptance in the building industry and is successfully employed in a large number of construction projects worldwide (Eastman et al., 2011).

During the planning of new buildings and infrastructure facilities, many decisions require integrated analysis of information regarding the new (planned) construction in combination with information about already existing structures. The required holistic data pool can be used to process queries that comprise semantic, spatial and temporal aspects. Thus, analyses regarding property situations, construction scheduling and geometrical conflicts can be performed. A number of researchers have developed methods to support such an integrated analysis by converting IFC data sets to CityGML. These approaches are discussed in more detail in Section 2. Because of the varying modelling methods and their distinct semantics, there is a high probability that information is lost during this transformation.

This paper presents the QL4BIM query language and its corresponding runtime environment, the QL4BIM system. The QL4BIM language supports integrated analysis of both IFC and CityGML models. Moreover, because no conversion between these two schemas is required, the loss of information caused by data transformations is avoided. The intermediate layer of QL4BIM is the integral part of the query language that abstracts from the analysed data schemas such as IFC and CityGML. Entities from the external schemas are encapsulated by the intermediate layer. Thereby, the query runtime environment can process these items. Although no conversion between the different external schemas is performed, com-

prehensive queries that operate simultaneously on instances of both schemas can be processed. As demonstrated by IFC and CityGML, QL4BIM is conceptually capable of analysing schemas that include semantic, spatial and temporal data by using generic operators. The benefit of the proposed approach is that CityGML and IFC can be temporarily brought together in one comprehensive (spatial) context. Therefore, only one query language is required and identical operators can be employed on both data sources. For example, integrated analysis beyond the extent of a single schema enables the detection of spatial conflicts between models of different scales.

The paper is organized as follows: Related work that addresses the combined processing and conversion of BIM and GIS data and existing query languages for BIM are discussed in Section 2. In Section 3, an overview of QL4BIM is given. Including the extension for geospatial processing, the components of the query runtime environment are discussed in Section 4. In Section 5, the QL4BIM data model is presented. This model acts as an intermediate layer that realizes the necessary abstraction of the IFC and the CityGML schema. Section 6 discusses the handling of CityGML geometry. In Section 7, a case study is shown. Here a combined IFC/CityGML data pool is analysed by two exemplary QL4BIM queries. Conclusions are presented in Section 8.

2 Related Work

The integration of approaches from geoinformatics and construction informatics has recently been addressed by a growing number of publications. In addition, the joint processing of data from these two domains is a topical subject. A general mapping between IFC and CityGML schemas for LOD1 to LOD4 has been developed by Isikdag and Zlatanova (2009). In Hijazi et al. (2011) the extension of city models with IFC data concerning utility networks is discussed. Topological analyses and spatial reasoning on both models were employed in Khan et al. (2014) to deduce indoor routing graphs. Their contribution also presents an approach to convert IFC models into CityGML LOD4 with a focus on navigable space. To integrate IFC data into a GIS context, the GeoBIM extension for CityGML can be used. Here an application domain extensions (ADE) is employed to equip the CityGML schema with 17 classes from the BIM domain (Laat and van Berlo, 2011). In El-Mekawy et al. (2011) a bi-directional mapping between IFC and CityGML was established using a combined data model. This requires a conversion to the combined model and to the respective data model. Conversely, the QL4BIM data model encapsulates items from IFC and CityGML without a bi-directional conversion operation. An automatic conversion of IFC data to CityGML LOD3 was also discussed in Donkers (2013). The spatial operators of QL4BIM are heavily influenced by spatial GIS analysis. Here, metric and topological analysis functionality which is common in GIS was adapted to the three-dimensional data of building information models (Daum and Borrmann, 2014) (Borrmann and Rank, 2009).

In addtion to QL4BIM, a number of query languages have been developed for the BIM domain. The partial model query language (PMQL) is an XML-based language offering *Select*, *Update* and *Delete* operators (Adachi, 2003). It has been developed in the context of the EXPRESS modelling languages which is used to define the IFC schema. However, PMQL does not provide high-level operators for geometric analysis. The building information model query language (BIMQL) is the query interface of the IFC model server (Mazairac and Beetz, 2013). The language is inspired by SPARQL which is a query language developed for the data model of the semantic web (Prud'Hommeaux et al., 2008). Although BIMQL offers an interface to extend the language with capabilities for spatial analysis, this is not yet available. A visual BIM query languages for craftsman was introduced in Wülfing et al. (2014). Finally, BERA is a language developed specifically to support code compliance checking (Lee, 2011).

3 Overview of QL4BIM

QL4BIM is an imperative and procedural query language for analysing and processing building information models. QL4BIM is designed to be employed by domain experts and offers a carefully selected vocabulary to formulate queries at a high level of abstraction. Only a few patterns must be understood to use the language. Low-level data handling operations such as instantiations and cast operations are hidden from the end user. Queries can be stated in textual and visual notation. In this contribution, the textual notation ^{t}QL4BIM is used to describe the underlying methodology. For a description of the visual notation ^{v}QL4BIM, the reader is referred to Daum and Borrmann (2015).

A complete QL4BIM query includes a number of statements executed successively. Each statement represents an assignment that binds the result of an operation to a variable. An operation is the combination of an operator and the passed parameters. Constants such as strings, numbers, floats as well as assigned variables can be used as parameters. In addition, user defined predicates that evaluate the attributes of entities can be handed over to operators. The described pattern is reflected by the textual grammar of QL4BIM that is depicted in extracts in Figure 1.

Listing 1 shows an example query stated in ^{t}QL4BIM that selects pairs of specific walls and windows, which are of the topological relationship *Touch*. The query begins by loading an IFC instance file using the *GetModel* operator. In Line 2, all walls are selected on the basis of their type affinity. The result is then filtered by applying a predicate to each entity. Thus, only walls with an attribute titled *Description* holding the value "Wall-002" are retained. To access an attribute of an entity, the point operator is employed in the predicate definition (Line 3). After retrieving all windows by applying a second *TypeFilter* (Line 4), the set of filtered walls and all windows are analysed topologically by applying the operator *Touch*. The result is assigned to a relational variable that comprises pairs of wall and window objects that satisfy a *Touch* relation.

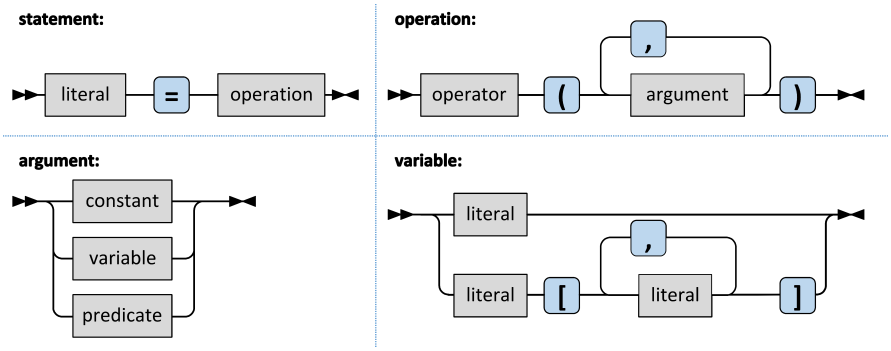

Fig. 1 Section of the QL4BIM textual grammar (syntax diagram)

```
model = GetModel("C:\Institute.ifc")
allWalls = TypeFilter(model, "IfcWall")
someWalls = AttributeFilter(allWalls.Description = "Wall-002")
allWindows = TypeFilter(model, "IfcWindow")
rel[wall, window] = Touch(someWalls, allWindows)
```

Listing 1 QL4BIM query extracting specific walls and their touching windows

This demonstrates the main pattern of QL4BIM, i.e. repeated variable assignment combined with the application of an operator. In this way, subsequent operations process variables assigned by former calls. Despite its simplicity, this pattern enables flexible and detailed analysis of building models by chaining manageable operations. The complexity of the information retrieval is hidden in the interior processing of each operator. Thus, the domain expert can concentrate on the semantic of each operator rather than define the low-level data handling. The functionality of the language is provided by four categories of operators:

1. Semantic operators for type and attribute extractions
2. Relational operators to analyse links between entities
3. Spatial operators to evaluate topological, directional and metric predicates
4. Temporal operators to examine construction schedules

Figure 2 demonstrates possibilities to select building elements in an IFC model instance using these four operator categories. For example, walls can be selected by evaluating their type affinity (1). To extract tuples of walls and their filling windows, the relations stored in the model are analysed (2). It is also possible to identify tuples of walls and their associated windows by a spatial examination. Here, the occurrence of *Touch* relationships between the geometric representations is used as a selection predicate (3). Finally, the selection of building elements can be based on their position in a construction schedule, e.g. the planned installation start time (4).

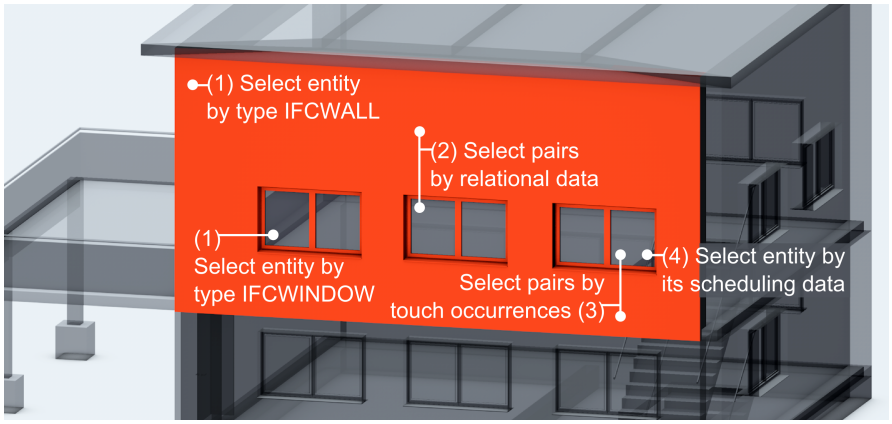

Fig. 2 Applications of the four categories of QL4BIM operators

4 QL4BIM's Runtime Environment: the QL4BIM System

The QL4BIM system operates using its own internal data model, which acts as an intermediate layer for abstraction and integrated analysis. This concept offers high flexibility with respect to the processed data. Instances of the data model of QL4BIM wrap entities of a genuine data model whereas IFC and CityGML are considered until now. In this contribution, types from QL4BIM will be addressed as *internal* whereas types from genuine data models are referred to as *external*. The QL4BIM data pool, which is the initial data source that can be analysed by queries, is provided in the internal schema. Furthermore, all intermediate variables are instances of this internal schema. To establish the data pool and to create an appropriate infrastructure for query execution, the runtime makes use of several parsers, a query interpreter and a query backend.

Figure 3 shows the components of the runtime environment and their interconnections. Items drawn in white are parts of the original system focused on IFC processing. The components newly developed to handle CityGML are shaded in blue. The included parsers are used for the syntactic analysis of query input, for importing the underlying pool data and for examining the external schema in use for the given case.

In the following, the interaction among components is discussed without the explicit consideration of the examined external data schema. As the internal data model of QL4BIM is highly generic, the principle is identical for handling IFC, CityGML or any other schema.

First, the instance parser interprets the referenced file and transfers the external entities to an internal representation. Rather than specific classes for external types, the runtime environment uses a late binding approach for this in-memory representation. As a result, entities can be set up dynamically by means of generic parts. Fine-grained sub-elements such as object identifiers, lists of symbolic references

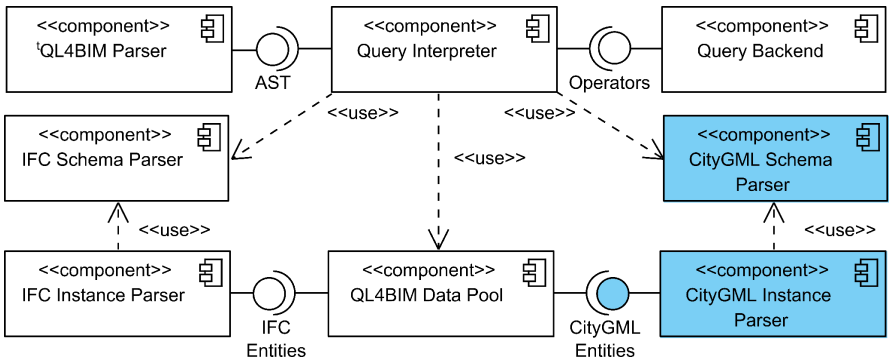

Fig. 3 Components of the QL4BIM system (UML component diagram)

and values of simple types are represented by these parts. The late binding approach provides a great flexibility regarding external schemas. Thus, all IFC versions are directly supported ad-hoc, and alternative data models such as CityGML can be integrated easily by an appropriate instance and schema parser.

The schema parser supports the query execution when external types must be compared against native QL4BIM types. In addition, meta-data such as the attribute names of entities can be read directly from the schema without computationally expensive reflection mechanisms.

If a query has passed the syntactic analysis, the tQL4BIM parser translates it to an abstract syntax tree (AST) (Parr, 2010). This intermediate representation of a query is consumed by the query interpreter. The AST is a streamlined structure deduced from the textual query input without lexical helpers. The tree structure can be processed in several ways. For example, the QL4BIM system performs validations on the AST that verify the order of variables. Additionally, static type checks are executed and parameter handovers are examined. Finally, the main purpose of the AST is its use for the query interpretation.

Figure 4 shows a QL4BIM query as an AST representation. The S-nodes correspond to the four sub-query statements. If a complete query execution is invoked by the user, the interpreter traverses the whole AST to collect results and to call the appropriate functions in the query backend. Finally, the last variable in the query includes the final output of the query.

As a matter of principle, the execution of a declarative query language such as SQL processes a complex query in a single step. In contrast, the QL4BIM system supports the execution of all statements and an incremental interpretation. The imperative style of the language combined with the enforced representation of interim results enables this incremental interpretation. Rather than processing all statements in the AST, the runtime environment interprets statements sequentially, according to user request. In this step-wise execution, the content of the current variable is visualized in the QL4BIM user interface. This approach is similar to a debugging session in an integrated development environment for a standard programming language.

The approach supports domain experts in formulating complex queries, because the effect of each single statement can be tested.

```
model =
   GetModel ("C:\Institute.ifc")

walls =
   TypeFilter (model, "IfcWall")

windows =
   TypeFilter (model, "IfcWindow")

r1[wall, window] =
    Touch (walls, windows)
```

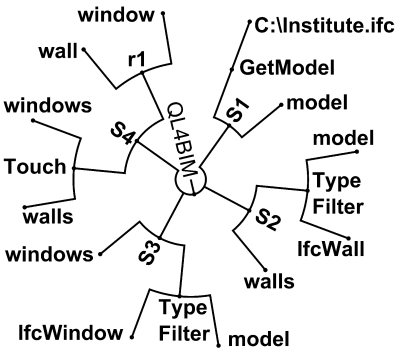

Fig. 4 A QL4BIM query and its AST-based representation

5 The QL4BIM Data Model

The central structure of the QL4BIM data model is the *Entity* which wraps an object supplied by an external data model. Instance files are imported into the data pool using the *GetModel* operator. After execution, the objects included in these files are present in the pool as a set of entities and are referenced by a variable. By executing QL4BIM queries, the initial sets are processed and intermediate results are bound to new variables. The processing includes the extraction of a single entity and the creation of variables that refer to specific sets and relations of entities.

To incorporate the original data, the *Entity* type is designed as a dynamic container for external data. Fine-grained elements below the level of a complete object are denoted as *Parts* in the QL4BIM system. The *Part* type accommodates primitive data such as strings, enumerations, integers and floats. Thus, an external object and its attributes can be represented in the runtime environment. Figure 5 shows a class diagram that depicts the *Entity* class and its association to the *Part* class. Furthermore, the composition of the internal *Set* and *Relation* classes are shown.

The query language is designed without explicit type declarations. Therefore, no *Entity*, *Set* and *Relation* keywords are used in QL4BIM queries (see Listing 1). A variable is defined by using a new literal in an assignment. The actual type of the variable is inferred by the operator used in this assignment. Figure 6 demonstrates this by annotations for the QL4BIM query in Listing 1.

Besides defined variables that bind to instances, sets or relations, the passing of constants as parameters is also depicted. The sub-elements of entities are processed to evaluate the *AttributeFilter*. Therefore, the point operator, which can be used in

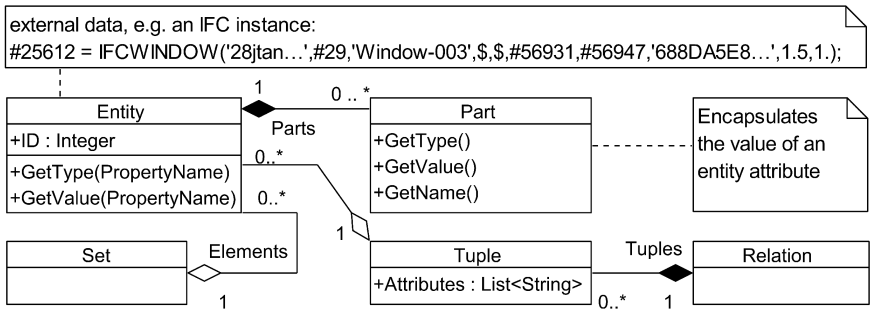

Fig. 5 Central classes of the internal QL4BIM data model (UML class diagram)

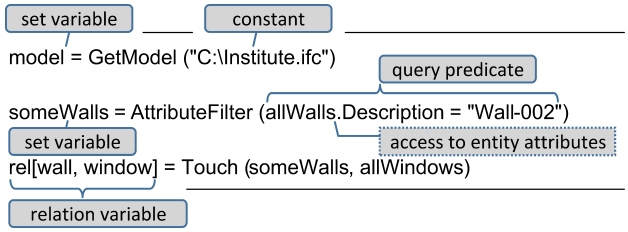

Fig. 6 Implicit typing of *Set* and *Relation* variables and query predicates

passed query predicates, is provided. In the example, each entity of the *allWalls* set is examined for an attribute named *Description*. If this attribute is present, its type is checked. Here, the attribute must be typed as a string as it is tested for equality against a string constant. After query execution, the *someWalls* variable will only include entities with a string typed attribute *Description* holding the value ”Wall-002”.

The processing of schemas of examined external data models is required to evaluate attribute names and types. Thus, the integration of any external data model in the QL4BIM system includes the introduction of an instance parser and a schema parser. To support CityGML, this is a parser for GML instance files and a parser for the respective XSD schema. The use of IFC data results in a parser for Part21 files and an EXPRESS schema parser.

Within the QL4BIM data model, the selection of entities is based on an unambiguous object identifier (OID). If an OID is available for any object in the original data, it is reused for this purpose. Otherwise, artificial OIDs are created by the QL4BIM runtime environment. The internal data model of QL4BIM is designed to encapsulate data from external models in a way that supports an efficient data processing. Therefore, internal entities and their corresponding attributes are created and stored in a data structure for random access. In addition, interrelations between entities represent further essential information. In the internal data model, these relations are expressed through foreign OIDs as the content of the respective *Part* of an *Entity* (Fig. 7). The data model of QL4BIM is designed to support this typed

graph structure. A hypothesis of this research is that representing entities and particularly their relations is required to analyse any data model in geospatial and BIM domains.

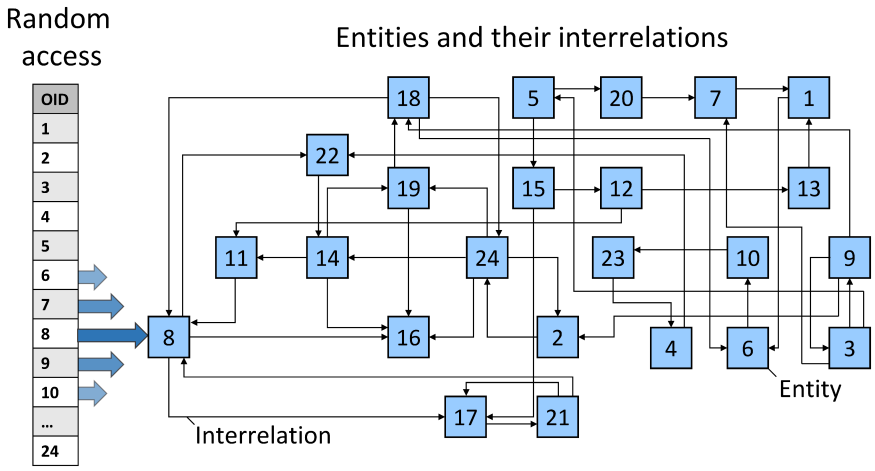

Fig. 7 Data structure for random access to entities and the resolving of interrelations

To realize the efficient selection of entities by their OIDs, the QL4BIM runtime environment introduces a hash table-based storage model. As a result, the selection of an entity and resolving an interrelation is executed on average in constant time, independent from the number of entities in the current input. Thus, the duration of query execution is reduced if large data sets are handled. This is crucial because a query language is generally applied if manual processing is not feasible.

The layout of a CityGML instance differs from the relational equipped linear structure of the QL4BIM runtime environment. As CityGML is encoded as XML, data is arranged in a hierarchical, tree-based fashion which is illustrated in Listing 2. In this encoding, an object is described by an XML element. Direct child elements represent the attributes of the object. If an attribute is modelled by a simple type, the actual value is encoded as the enclosed text of the element. Accordingly, the *bldg:Building* instance in Listing 2 holds a *gml:Name* attribute with the value ”Example Building LOD4”. If an attribute references another object it is encoded as a nested XML element. This is the case for the building with the *gml:boundedBy* attribute which points to a *bldg:GroundSurface* instance. Additionally, *XLink* references can be used to link to objects at other places in the document.

```
<bldg:Building gml:id="GML_7...">
  <gml:name>Example Building LOD4 </gml:name>
  <bldg:measuredHeight uom="#m">5.0</bldg:measuredHeight>
  <bldg:storeysAboveGround>1</bldg:storeysAboveGround>
  <bldg:storeyHeightsAboveGround uom="#m">3.0</bldg:storey...>
  <bldg:boundedBy>
    <bldg:GroundSurface>
      <gml:name>Ground Slab</gml:name>
      <bldg:lod4MultiSurface>
        <gml:MultiSurface> ...
        </gml:MultiSurface>
      </bldg:lod4MultiSurface>
    </bldg:GroundSurface>
  </bldg:boundedBy>
```

Listing 2 Tree-based encoding of a CityGML instance

If CityGML instances are parsed and imported into the QL4BIM data model, the nested data structure must be linearised so that CityGML objects can be represented in the hash table-based data pool. The rule for this transformation is that every object is transferred to a QL4BIM *Entity* and its attributes are represented as *Part* instances. If an attribute of an objects is a simple type, it can be stored directly in a *Part*. If another object is referenced in an attribute, a stand-alone *Entity* is instantiated and inserted into the indexed data pool. The linkage between the two entities is expressed by the OID value stored in the attribute of the parent. In contrast to a conversion between CityGML and IFC, the embedding of the external objects in the QL4BIM data model is bijective. This is evident as the imported data can be re-encoded to its native format.

As CityGML is encoded in XML, namespaces are frequently used in the data model. To achieve complete representation of the CityGML data, namespace values are added as meta data to QL4BIM entities and their parts. Furthermore, XML elements can be equipped with attributes. Although, attributes are not heavily used in CityGML, their representation is useful. For example they provide information about the unit used for stated quantities. Thus, XML attributes are upgraded to regular attributes of entities and can be consistently analysed in queries. The main parts of the IFC and CityGML models can be combined in the internal QL4BIM data model without special handling of the respective source. This results in identical queries in many cases. Listing 3 shows the extraction of IFC and CityGML walls whose *Name* attribute ends with ”South”.

```
ifcModel = GetModel ("C:\Institute.ifc")
ifcWalls = TypeFilter (ifcModel , "IfcWall")
someIfcWalls = AttributeFilter (ifcWalls.Name = "*South")
gmlModel = GetModel ("C:\Building_LOD4.gml")
gmlWalls = TypeFilter (gmlModel, "WallSurface")
someGmlWalls = AttributeFilter (gmlWalls.name = "*South")
```

Listing 3 Identical queries for IFC and CityGML data

6 Handling of Surface-oriented CityGML Geometry

The spatial operators of QL4BIM examine the geometry representation of entities in the data pool. In the IFC case, the used geometry representation is based on solids that have a well-defined interior, boundary and exterior (Daum and Borrmann, 2014). As CityGML modelling is designed to represent geometry captured by survey, laser scanning or photogrammetry, the use of surface-oriented, non-closed representations is common. Figure 8 shows non-closed geometry in an exemplary CityGML data set in combination with IFC flow segments which are located below the CityGML entities.

A spatial analysis is also possible if the surface-oriented geometry of CityGML is used together with IFC solids. However, because of the lacking interior/exterior classifications within the surface representation, e.g. directional constellations can remain undetected. Here, the wall surfaces can not be classified as laying above the flow segments because of the missing covers of walls. To cope with this, the spatial analysis should be based on a extended set of geometry representations and include the *GroundSurface* instance.

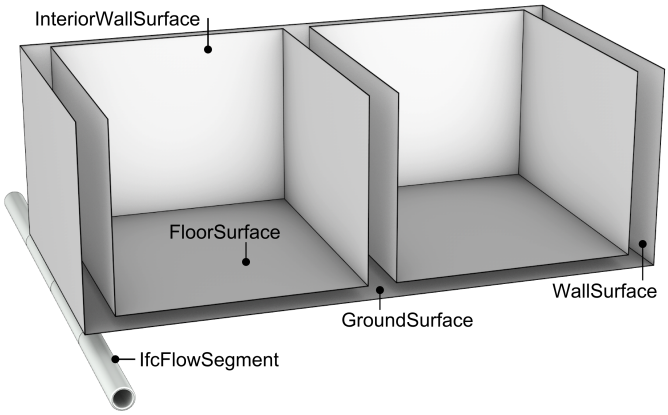

Fig. 8 Non-closed geometry in an exemplary CityGML data set in combination with IFC solids

7 Case Study

The following case study demonstrates the concept of an integrated analysis of data from the domains of construction informatics and geoinformatics. The study examines the interactions of subterrestrial constructions that are part of a planned subway track with existing buildings on the surface. Here, the buildings above ground are

present as inventory data expressed in CityGML. The planned subway track and its supporting buildings are modelled for construction purpose and are available as an IFC data set. In Figure 9, the CityGML entities are shaded in grey and IFC entities are shaded in blue.

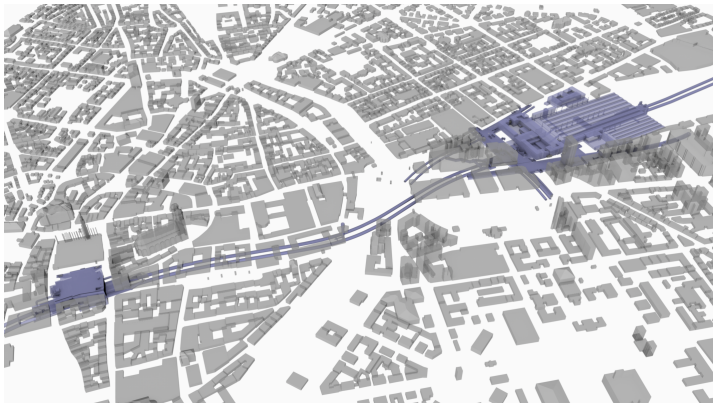

Fig. 9 CityGML data set of Munich (LOD1) combined with a planned subway track encoded as IFC

The following QL4BIM query in Listing 4 begins by loading the city model into the runtime environment. Then, a selection of buildings is executed on the basis of their type affinity. The same is done with the IFC model from which all spaces are retrieved. The final *Above* operator tests which buildings are located above spaces instances. Thus, the relational *rel[space, building]* variable represents pairs of *IfcSpace* and CityGML *Building* objects.

```
1  gmlModel = GetModel ("C:\Building_LOD1.gml")
2  gmlBuildings = TypeFilter (gmlModel, "building")
3  ifcModel = GetModel ("C:\supportinBuildings.ifc")
4  ifcSpaces = TypeFilter (ifcModel , "IfcSpaces")
5  rel[space, building] = Above(ifcSpaces, gmlbuildings)
```

Listing 4 QL4BIM query spatially analysing IFC spaces and CityGML buildings

```
1  gmlModel = GetModel ("C:\Building_LOD4.gml")
2  buildings = TypeFilter (gmlModel, "building")
3  highBuildings = AttributeFilter(buildings.StoreysAboveGround > 3)
4  walls = RelationResolver(highBuildings.WallSurface)
5  ifcModel = GetModel ("C:\supportinBuildings.ifc")
6  flowSegments = TypeFilter (ifcModel , "IfcFlowSegment")
7  rel[flowSegment, wall] = NearerThan(flowSegments, walls, 3)
```

Listing 5 QL4BIM query analysing IFC flow segments and walls in CityGML buildings

Listing 5 shows a second example of combined IFC/CityGML processing, demonstrating a more detailed analysis. From the loaded CityGML data, only buildings

that are higher than three storeys above ground, are considered. In the next step, walls are selected from these buildings. Then, the set of CityGML walls is examined spatially with a set of flow segments originating from the IFC data set. Finally, a relation is established that contains pairs of one IfcFlowSegment and one CityGML WallSurface whereby the two entities are located closer than three metres.

8 Conclusion

This paper present an extended version of the QL4BIM query language that supports integrated processing of data sets from the AEC and the geospatial domains. Rather than converting between the data models of IFC and CityGML, an intermediate layer as part of the QL4BIM system is introduced. As a result, it is possible to abstract from different modelling approaches. The original data remains valid and integrated data processing is realized. In essence, a new level of integrated semantic and spatial analysis across domain borders of BIM and city models is achieved. Thus, the proposed approach has the potential to ease complex urban and structural planning substantially. Future work will concentrate on the spatial processing of surface-oriented geometry of CityGML in combination with the solid representations of IFC. Furthermore, the spatial referencing in the QL4BIM System will be discussed.

References

Adachi, Y. (2003). Overview of partial model query language. In *ISPE CE*, pp. 549–555.

Borrmann, A., T. H. Kolbe, A. Donaubauer, H. Steuer, J. R. Jubierre, and M. Flurl (2015). Multi-scale geometric-semantic modeling of shield tunnels for GIS and BIM applications. *Computer-Aided Civil and Infrastructure Engineering 30*(4), 263–281.

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

Daum, S. and A. Borrmann (2014). Processing of Topological BIM Queries using Boundary Representation Based Methods. *Advanced Engineering Informatics 28*(4), 272–286.

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

Donkers, S. (2013). *Automatic generation of CityGML LoD3 building models from IFC models*. Ph. D. thesis, TU Delft, Delft University of Technology.

Eastman, C., P. Teicholz, R. Sacks, and K. Liston (2011). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*, Volume 2.

El-Mekawy, M., A. Östman, and K. Shahzad (2011). Towards interoperating cityGML and IFC building models: a unified model based approach. In *Advances in 3D Geo-Information Sciences*, pp. 73–93. Springer.

Gröger, G. and L. Plümer (2014). The Interoperable Building Model of the European Union. In A. Abdul Rahman, P. Boguslawski, F. Anton, M. N. Said, and K. M. Omar (Eds.), *Geoinformation for Informed Decisions*, Lecture Notes in Geoinformation and Cartography, pp. 1–17. Springer International Publishing.

Hijazi, I., M. Ehlers, S. Zlatanova, T. Becker, and L. van Berlo (2011). Initial investigations for modeling interior Utilities within 3D Geo Context: Transforming IFC-interior utility to CityGML/UtilityNetworkADE. In *Advances in 3D Geoinformation sciences*, pp. 95–113. Springer.

Isikdag, U. and S. Zlatanova (2009). Towards defining a framework for automatic generation of buildings in CityGML using building Information Models. In *3D Geo-Information Sciences*, pp. 79–96. Springer.

Khan, A. A., A. Donaubauer, and T. H. Kolbe (2014). A multi-step transformation process for automatically generating indoor routing graphs from existing semantic 3D building models. In *Proceedings of the 9th 3D GeoInfo Conference*.

Kolbe, T. H. (2009). Representing and exchanging 3D city models with CityGML. In *3D geo-information sciences*, pp. 15–31. Springer.

Laat, R. d. and L. van Berlo (2011). Integration of BIM and GIS: The development of the CityGML GeoBIM extension. In *Advances in 3D Geo-Information Sciences*, pp. 211–225. Springer.

Lee, J. K. (2011). Building environment rule and analysis (BERA) language and its application for evaluating building circulation and spatial program.

Mazairac, W. and J. Beetz (2013). BIMQL – An open query language for building information models. *Advanced Engineering Informatics 27*(4), 444–456.

Parr, T. (2010). *Language implementation patterns: Create your own domain-specific and general programming languages*. The pragmatic programmers. Raleigh, N.C.: Pragmatic Bookshelf.

Prud'Hommeaux, E., A. Seaborne, et al. (2008). SPARQL query language for RDF. *W3C recommendation 15*.

Wülfing, A., R. Windisch, and R. J. Scherer (2014). A visual BIM query language. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014*, 157.