A MULTI-PERSPECTIVE APPROACH TO INTERPRETING SPATIO-SEMANTIC CHANGES OF LARGE 3D CITY MODELS IN CITYGML USING A GRAPH DATABASE

Son H. Nguyen[1,*], Thomas H. Kolbe[1]

[1] Chair of Geoinformatics, Department of Aerospace and Geodesy, Technical University of Munich (TUM), Germany - (son.nguyen, thomas.kolbe)@tum.de

KEY WORDS: CityGML, Spatio-semantic Comparison, Change Detection, Edit Operations, Graph Database, Analysis

ABSTRACT:

In the age of virtualization, rapid urbanization and fierce competition, more and more "digital twins" of real cities are being created as a time, cost-efficient and especially user-oriented solution to many problems in urban planning and management. One prominent task is to efficiently detect progresses made by a city based on their virtual 3D city models recorded over the years, and then interpret them accordingly with respect to different groups of users and stakeholders involved in the process. The first half of the problem, namely automated change detection in city models, has been addressed in recent studies. The other half of the problem however, namely a user-oriented interpretation of detected changes, still remains. Thus, based on the current findings, this research extends the conceptual models and definition of different types of edit operations between city models using a graph database, where the graph representations of city models are also stored. New rules and conditions are then provided to further categorize these changes based on their semantic contents. Considering the different expectations and requirements of different groups of users and stakeholders, the research aims to provide a multi-perspective interpretation of such categorized changes.

1. INTRODUCTION

In the age of virtualization, rapid urbanization and fierce competition, more and more "digital twins" of real cities are being created and maintained not only by private companies and sectors, but also by numerous cities and countries all around the world. Managing digital twins should be time and cost-efficient and especially user-oriented, since a number of different users and stakeholders that have different interests and expectations are often involved in the process. As a result, one often-stated problem is how to automatically detect progresses made by a city based on its virtual city models recorded over the years. This should be done not only in a time and cost-efficient but also in a user-oriented manner. This question is interesting both due to its technical challenges and many possible interpretations of detected changes with respect to different groups of users and stakeholders.

Progress or change detection in virtual semantic 3D city models (mostly encoded in CityGML) is not new but due to its technical difficulties, only a few studies have been published so far, such as by (Bakillah et al., 2009) and (Redweik and Becker, 2015). The technical challenge of this problem can basically be explained by understanding the key concepts and characteristics of CityGML itself. Firstly, CityGML is an official open standard for the storage and exchange of virtual 3D city models. The OGC standard is capable of describing most common urban objects such as buildings, bridges, tunnels, water bodies, vegetation, traffic, etc. Application areas of CityGML vary widely ranging from urban planning and architectural design to environmental and traffic simulation (Gröger et al., 2012). Secondly, in contrast to other conventional virtual 3D city models that are purely graphical or geometrical, CityGML combines both spatial and semantic information in one place. This enables not only object visualization, but also analysis of the enriched thematic data. Thirdly, CityGML represents 3D city objects in five different levels of details (LODs 0 - 4) capable of describing different geometric details of complex objects. And finally, CityGML is known (and also debated) for its high level of syntactic flexibility by allowing multiple syntactic representations of the same object. For instance, boundary surfaces of a solid can either be defined "in-line" directly, or referenced to existing surfaces that are e.g. shared with other neighbouring solids using their identifiers (i.e. XLinks). Additionally, each surface can either be defined as a single polygon or as a composite surface consisting of smaller surface patches.

In a recent study, (Nguyen et al., 2017) addressed the posed complexities and technical difficulties in detecting changes between CityGML datasets by considering them as graphs due to the graph nature of CityGML datasets. The authors then provided both a detailed concept and a working open-source implementation of how to map CityGML datasets onto graphs as well as how to match them using a graph database.

The work presented by (Nguyen et al., 2017) however only solved the first half of the problem, namely the automated approach to change detection between virtual city models. The second half of the problem, namely to provide a multi-perspective interpretation of detected changes with respect to different groups of users and stakeholders, still remains. Expectations and interests vary vastly among users and stakeholders. For instance, developers and programmers are interested in every detected change between the city models, from updated identifiers of polygons, lines, etc. to deletion of entire buildings. Surveyors and data administrators are mostly concerned about how to update and maintain 3D city models continuously and efficiently. In contrast, municipalities and city administrators are rather interested in the large-scale progresses that occurred in the city, including changes of top-level features such as the number of buildings with "real" detected changes, the number of buildings that are "unchanged", and what the most frequent changes are, etc.

Therefore, although equally challenging, this part of the problem requires an understanding of how detected changes are represented in the graph database as well as of how relevant each of these

*Corresponding author

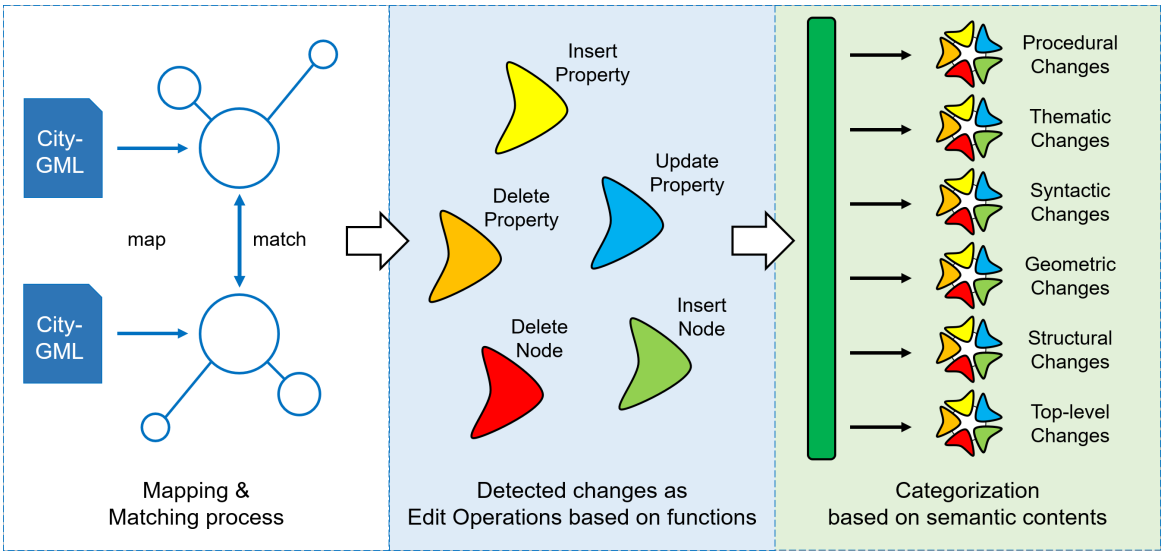


Figure 1. An overview of the workflow presented in this research. Based on the concepts and implementation proposed by (Nguyen et al., 2017), the mapping and matching process are extended to better produce edit operations from detected changes. Each coloured petal in the middle stage represents a type of edit operations. Each category in the last stage can therefore contain all five types of edit operations.

changes is to the stakeholders. This plays a vital role in connecting the technical implementation with users and stakeholders in the real world. Achieving this may open up new opportunities and possibilities on a better understanding of cities' evolution and developing new strategies and ideas accordingly.

Thus, this paper proposes: 1) enhancing the mapping and matching process of CityGML datasets based on the methods and implementation provided by (Nguyen et al., 2017) to enable complex analysis and querying of detected changes, 2) extending the conceptual models and definition of edit operations representing such detected changes, and 3) developing new methods and rules to further categorize edit operations in a user-oriented way. An overview of this workflow is illustrated in Figure 1. Section 2 discusses studies and research work that are most relevant to the concepts and methods proposed in this paper. Section 3 covers the first two steps of the above-mentioned workflow, while Section 4 explains the last step. Then, Section 5 introduces and classifies users and stakeholders in different groups and shows the relationships between them and the different categories of changes. In Section 6, test results and findings are presented. Finally, Section 7 concludes the paper and discusses some future work.

2. RELATED WORK

Most virtual 3D city models are encoded in CityGML, which is an application schema of the Geography Markup Language 3 (GML3). GML is an XML application for expressing spatial and geographical data issued by the Open Geospatial Consortium (OGC) (Gröger et al., 2012). CityGML datasets are therefore stored as text documents. In this regard, conventional *diff* tools such as the Hunt–Szymanski algorithm (Hunt and Szymanski, 1977) and Myers' algorithm (Myers, 1986) are well-known for their capability of comparing and displaying differences between two plain text files. However, despite being text files, CityGML datasets contain highly structured information and thus cannot simply be compared using conventional *diff* tools designed only for plain texts.

Since XML documents can be conceptually interpreted as a tree data structure (i.e. XML tree), (Redweik and Becker, 2015) proposed an approach to detecting changes between CityGML datasets using their tree representation. The authors employed the algorithm "X-Diff" (Chawathe et al., 1996; Wang et al., 2003), which could compare two unordered XML trees using standard tree-to-tree corrections. One major advantage of the methods proposed by (Redweik and Becker, 2015) is that they considered both the geometric and semantic information available in CityGML documents. However, due to the fact that CityGML allows the usage of XLinks, which enable linking between elements that could form potential cycles or cause a node to have multiple parents, CityGML documents are generally not structured as a tree, but as a cyclic graph instead. Thus, due to the graph nature of CityGML datasets, this approach is often not expressive enough.

(Falkowski and Ebert, 2009) presented a graph-based schema for integrated models of urban data in CityGML using the TGraph technology. They defined an explicit graph representation of geometric, topological, semantic and appearance objects and showed how they can be stored in one integrated graph model. The authors explained the vital role of such graph representations in various efficient processing algorithms, including model creation, improvement, transformation, analysis and export of city objects. In another relevant research, (Agoub et al., 2016) showed limitations of storing and managing highly complex data structures such as CityGML in a relational database management system (RDBMS) such as PostgreSQL/PostGIS or Oracle Spatial. They argued that mapping object-oriented data models with well-defined objects, attributes and relations into compact relational schemas without causing information loss is a challenging task. The research suggested employing a NoSQL database, particularly a graph database such as Neo4j, to "help dealing with this problem as the underlying data model of nodes and edges can natively represent a conceptual UML diagram". Both the concepts provided by (Falkowski and Ebert, 2009) and (Agoub et al., 2016) were promising as they showed the potential of using graphs to represent highly complex hierarchical objects in CityGML. However, their methods were either rather a proof of concept or light-weight. They did not explain in details how hierarchical information (like inheritance) and cross-referencing (as in XLinks) of CityGML objects can be fully and automatically mapped onto graphs without losing data during the process.

In this context, (Nguyen et al., 2017) proposed an approach to mapping, matching and updating CityGML datasets using a graph database. By taking advantage of the available inheritance relations defined in the object-oriented model, the presented mapping methods were able to transform complex hierarchical CityGML objects fully onto corresponding graph components with no information loss. XLink references between objects were also taken into account and reflected in the resulting graphs. Then, based on the mapped graphs, the matching progress ensured only the most likely sub-graph candidates are compared based on their e.g. thematic, geometric or semantic information. Detected changes were also represented as graph nodes attached to the sources and could be applied to generate edit operations and queries to update the old CityGML datasets using e.g. the Web Feature Service (WFS).

Furthermore, the above-mentioned papers such as (Chawathe et al., 1996; Wang et al., 2003; Redweik and Becker, 2015) and (Nguyen et al., 2017) provided a classification of detected changes as the so-called edit operations. For instance, (Redweik and Becker, 2015) divided them into three groups: inserting a child node to a parent node, deleting an existing node from the tree and updating the value of a leaf node. On the other hand, (Nguyen et al., 2017) classified the detected changes as five edit operations: inserting a child node or a sub-graph to a parent node or graph, deleting an existing child node or a sub-graph from the database, inserting a new simple property (that can be stored as texts) to a node, deleting an existing simple property from a node, and updating the old value of a simple property with a new one. However, these studies did not further explain how such edit operations can be interpreted. They left the question open as how to further classify edit operations based on not only their functionalities in the underlying concepts, but also how relevant they were to specific groups of users.

Thus, this research further focuses on filling this gap between technical realization of change detection concepts and how to better interpret them for different groups of users and stakeholders. Since the work presented in (Nguyen et al., 2017) was one of the first that actually provided a detailed concept of mapping and matching CityGML datasets using graphs as well as a working open-source implementation that scales well with very large data sizes, the implementation of this study was based on and extended from the findings of that work.

3. EXTENDING AND MANAGING EDIT OPERATIONS IN THE GRAPH DATABASE

In this section, the detected changes are classified as five types of edit operations based on their functionalities. These classes are $InsertProperty$, $DeleteProperty$, $UpdateProperty$, $InsertNode$ and $DeleteNode$.

3.1 Extended Definition and Conceptual Model

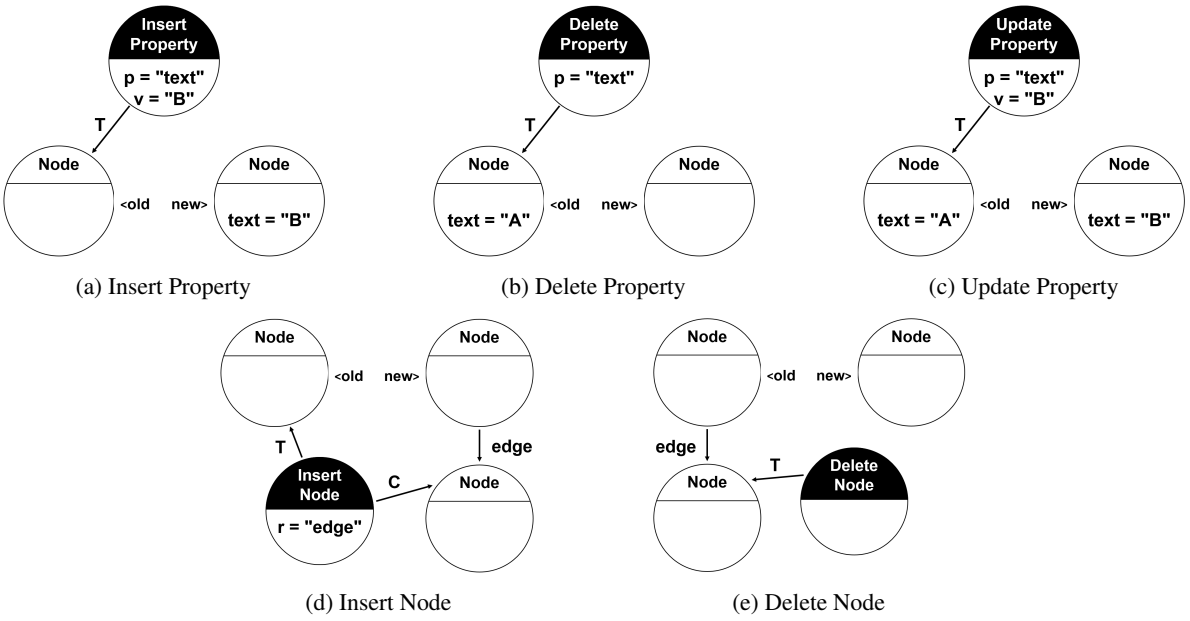
To compare two arbitrarily large CityGML datasets, a graph database such as Neo4j can be employed, where both the mapping and matching process take place. In Neo4j, a graph contains a set of nodes and edges. Each node and edge can have an arbitrary number of simple properties (e.g. texts, numbers, dates, etc.), whose names must however be unique within that node or edge. For any detected change, a node is created and attached to the source via an edge in the graph database on the fly every time a deviation is found. Conceptually, these changes are divided into five different types called edit operations. (Nguyen et al., 2017) provided a hierarchical modelling of such edit operations but did not explicitly define how they could be manipulated for complex queries in the graph database. Since edit operations play a central role in analysing and understanding the evolution of cities, this research proposes an extension of the definition and classification of these edit operations.

The five types of edit operations operate on two levels described as follows:

- Property level (representing edit operations on simple values):
  - $insert(p, v, T)$: insert a simple property p into the target node T with the value v;
  - $delete(p, T)$: delete the simple property p from the target node T;
  - $update(p, v, T)$: replace the old value of the simple property p from the target node T with new value v.
- Node level (representing edit operations on complex objects):
  - $insert(C, T, r)$: insert a child node C to the target node T with an edge or relation r between the nodes (in direction from T to C);
  - $delete(T)$: delete the target node T.

Note that the child node C and target node T can be either a leaf node or a root node representing a sub-graph (in the sense that the whole sub-graph is reachable from this root node). Since edit operations are generated with the older city model as the basis, the target node T always points to a node from the older dataset. Moreover, with the exception of edit operation nodes, neither new nodes are created nor existing nodes are deleted from the database. Both C and T are solely references pointing to an already existing node in the graph database (either from the older or newer city model). These pointers can be thought of as the tags "to be inserted", "to be deleted" or "to be updated" attached to the detected changes. Figure 2 gives an overview of how these edit operations are stored in the graph database.

The above-mentioned parameters such as p, v, C, T are required to define edit operations. However, to further allow their categorization and interpretation in a meaningful way (as presented in Section 4), some additional information may be needed. For instance, a unique ID id shall now be assigned for each edit operation. This can be the node ID assigned in the graph database or regenerated using customized patterns. Furthermore, the property $topLevelId$ indicating the GMLID of the top-level feature (such as building), to which the target node T belongs, shall also be stored in each edit operation. This particular useful piece of information allows efficient grouping of edit operations per top-level feature later on in Section 4. The value $topLevelId$ can be determined by traversing starting from the node T upwards in the graph until a node representing a top-level feature (e.g. a building or a road feature) has been reached. In Neo4j, although edges are directional, they can be traversed in both directions. Finally, the node type (or label) $TType$ of the node T also plays an important role in categorizing changes and shall be added to each edit operation. Although $TType$ can easily be retrieved by querying the type of node T, not all applications have direct access to the graph database. Thus, storing $TType$ in an additional simple text value ensures this information can also be shared to other tools outside of the graph database. The same can be applied to C and its node type $CType$.



(a) Insert Property (b) Delete Property (c) Update Property

(d) Insert Node (e) Delete Node

Figure 2. An illustration of how the five different types of edit operations (circles with filled label) are represented in the graph database. Each empty circle represents a node in the graph representation of the older (left) or newer (right) city model.

Therefore, based on the UML class diagram given in (Nguyen et al., 2017), this research further extended it with the above-mentioned modifications shown in Figure 3.

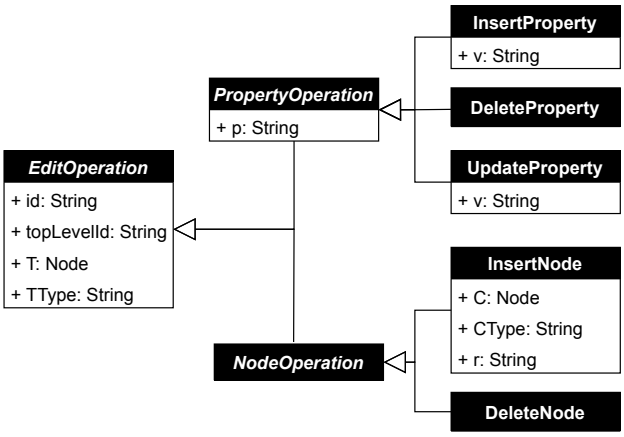


Figure 3. An extended UML class diagram of all edit operations.

3.2 Querying Edit Operations in the Graph Database

In order to allow efficient queries on the created edit operations on a graph level, some tweaks may be needed. In Neo4j for instance, to reduce the number of unnecessary database hits caused by each query, unique indexes are created for all edit operation nodes. Alternatively, a single unique indexed node named e.g. *RootMatcher* (which is not shown in Figure 3) can be created and connected to all edit operation nodes via the relationship *contains*. This allows fast and efficient queries on all edit operations. For example, using Cypher (the official graph query language used in Neo4j), Listing 1 and 2 show how the total number of all generated *InsertNode* operations can be calculated.

```
MATCH (i:INSERT_NODE)
RETURN COUNT(*)
```

Listing 1. An example Cypher query for counting all generated `INSERT_NODE` operations. Here all edit operations are indexed.

```
MATCH (m:ROOT_MATCHER)
RETURN SIZE((m)-[:contains]->(i:INSERT_NODE))
```

Listing 2. An example Cypher query for counting all generated `INSERT_NODE` operations. Here a single indexed root node is used to retrieve all connected edit operations.

3.3 Exporting Edit Operations

As mentioned before, not all applications have direct access to the graph database. As a result, the generated edit operations and their information should be exported to some common exchange formats that other programs can make sense of. For instance, tables can be used to provide a relational overview of all edit operations and thus are more attractive towards most Relational Database Management Systems (RDBMS). Comma-Separated Values (CSV) tables are a good way to capture some important data extracted from the edit operations stored in the graph database (see Table 1). These include all basic property values defined in Figure 3 (except the complex node objects C and T, which shall then be replaced by the corresponding text values $CType$ and $TType$ mentioned before in Section 3.1). This suffices for edit operations on the property level. Edit operations on the node level like *InsertNode* and *DeleteNode* are however more difficult to handle. In fact, if C and T refer to graph representations of complex objects, it is impossible to store such data as plain texts without losing information. Therefore, tables can represent some information of the edit operations stored in the graph database, but not completely. Nonetheless, they should provide sufficient data for most analysis and statistics processes on the metadata level.

ID	Top-level ID	T-Type	C-Type	Relationship
1	BLDG_1	Building	Bounding-Shape	bounded-By
...	...	...	...	...

Table 1. An example of a table containing information exported from all *InsertNode* operations stored in the graph database. Each table row represents an edit operation.

4. CATEGORIZING DETECTED CHANGES BASED ON THEIR SEMANTIC CONTENTS

Section 3 extended the definition and classification of detected changes as five classes of edit operations based on their functionalities. These are required to e.g. update the older city model to a newer one. However, such edit operations focus on the literal contents and do not explicitly differentiate between the semantic contents stored within the detected changes. These semantic contents are needed to enable detailed analyses on detected changes. Moreover, to the majority of users, it is difficult to identify "real" changes from e.g. syntactic changes in an often overwhelming number of generated edit operations. Hence, this section shall address these problems and explain how to further categorize changes in a more user-oriented manner. Based on the semantic contents, the changes detected by the change detection tool can further be divided into the following categories: procedural, thematic, (purely) syntactic, complex geometric, structural, and top-level changes. Most of the element and class names used in this section are defined in the CityGML specification (Gröger et al., 2012).

4.1 Procedural Changes

Modifications that were made due to change in programs, methods and procedures while handling CityGML datasets (e.g. import or export) are categorized as "procedural changes". These include:

- Changes in identifiers (e.g. those of top-level features such as buildings, bridges, tunnels, etc. as well as geometric elements such as polygons, lines, etc.);
- Changes in *creationDate* and *terminationDate*;
- Changes related to coordinate reference systems, such as *srsName*, *srsDimension*, etc.

Procedural changes mostly occur in simple properties and hence can be identified by comparing the property name p in each edit operation (see definition in Section 3.1) with the property names listed above. The value p can be retrieved either by using simple queries on the property names p similar to those shown in Section 3.2, or by searching for the values in the column p in each table exported in Section 3.3. The same also applies for the categorization of other types of changes in the next sections, except when a direct access to the graph database is required (as in the case of syntactic and geometric changes in Section 4.3 and 4.4).

4.2 Thematic Changes

One major advantage of CityGML is that it also includes thematic data in its encoding. Such can be stored as plain texts or numeric data. Note that due to limited space, not all properties or elements defined in CityGML are listed here.

4.2.1 Thematic Changes in Text Data Thematic text elements in CityGML are listed as follows:

- Simple text properties defined in CityGML such as *class*, *function*, *roofType*, etc.;
- Simple text properties defined in GML such as *location*, *description*, *name*, etc.;
- Generic attributes *stringAttribute* and *uriAttribute*;
- Properties and objects used to store address of the city object such as *country*, *postalCode*, *thoroughfare*, etc.;
- Properties and objects related to the appearance of the city object such as *imageUri*, *theme*, etc.

Note that, according to the mapping rules, while most simple thematic properties are stored directly as a text property within their parent node (which is often a node representation of a top-level feature, such as in the case of the property *class* in a building node), some other thematic objects such as *address* and *appearance* are more complex and cannot be represented by a single text, but a series of nodes or a sub-graph. However, regardless of their complexity, such thematic elements can always be represented by a limited number of text nodes and are thus all considered as thematic text data. Therefore, an edit operation is classified as "thematic text change", if its property name p is equal to one of the simple thematic text property names, or if its node representation is attached to a sub-graph belonging to one of the more complex thematic elements listed above.

4.2.2 Thematic Changes in Numeric Data The following list contains numeric thematic elements defined in CityGML:

- Simple numbers containing natural or real values such as *storeysAboveGround*, *storeysBelowGround*, *storeyHeightsAboveGround*, *storeyHeightsBelowGround*, *relativeToTerrain*, *relativeToWater*, etc. as well as generic *intAttribute* and *realAttribute*;
- Numeric values with explicit unit of measurement such as *measuredHeight*;
- Date values such as *yearOfConstruction*, *yearOfDemolition*, etc. and generic *dateAttribute*.

Such thematic elements should be handled numerically. This means that e.g. only when the absolute difference between two numbers exceeds a certain threshold or an error tolerance, an edit operation shall then be created. This also must take units of measurement and date values (if available) into account. Numeric changes that are too small (i.e. within an error tolerance) shall be considered as (purely) syntactic changes instead (see Section 4.3).

4.3 (Purely) Syntactic Changes

The same object sometimes can be represented in different ways. This is due to the syntactic ambiguity allowed in (City)GML. These changes are categorized as "(purely) syntactic changes" in this section, which means that only the syntactic representation of an object has been changed, while the object is semantically considered the same. Such changes are flagged as optional meaning that the corresponding edit operations are not required to be executed while updating. This includes syntactic ambiguities allowed in XML such as:

- Changed order of sibling elements of the same XML parent node, this is already taken into account while reading CityGML input datasets and shall not be further considered;
- Objects that are defined "in-line" or by using XLinks to link to other existing elements (e.g. a Solid can have a list of XLinks referring to other surfaces already declared somewhere in the dataset), this was also already solved by representing CityGML documents as graphs and shall not be further considered.

In addition, different syntactic representations of geometrically equivalent elements are also considered as (purely) syntactic changes. Such geometric elements include:

- **Points**: Two Points are considered geometrically equivalent if their coordinates are given within a small error or distance tolerance. Points can also be represented in many different syntactic ways, such as using *Point*, *Coord*, *Coordinates*, etc.

- **Line Segments** (*LineStrings*): Two *LineStrings* are considered geometrically equivalent if their control points are also geometrically equivalent, where no three consecutive control points are collinear. This can be extended for *Curves*.
- **3D Rings** (*LinearRings*): A *LinearRing* can be thought of as a closed *LineString* (here only co-planar control points are considered), thus two *LinearRings* are considered geometrically equivalent if the shapes bounded by their *LineStrings* are geometrically equivalent. This is true if the two shapes contain each other's control points within an error or distance tolerance.
- **Polygons**: Two polygons are considered geometrically equivalent if the *LinearRings* bounded by their exteriors and interiors are also geometrically equivalent. This can be extended for *Surfaces*, *MultiSurfaces*, etc.
- **Solids**: Two solids can be matched using their polygon footprint or their minimum bounding box. In the latter case, two minimum bounding boxes are considered potentially matched, if the ratio of their overlapping volume over their volume satisfies a reasonable lower limit, preferably close to 1. However, since different 3D objects could have the same minimum bounding box, solids matched by this method need to be further compared by successively matching their boundary polygons.

In contrast to simple thematic properties, identifying syntactic changes requires direct access to the graph database in order to retrieve their entire graph representations.

4.4 Complex Geometric Changes

Changes in geometric objects that are not purely syntactic (as described in Section 4.3) shall be categorized as "complex geometric changes". In contrast to (purely) syntactic changes that are optional, complex geometric ones are "real changes" (such as an enlargement of wall, door and window surfaces due to building renovation) and must be considered. These include:

- *lodXSolid*, *lodXTerrainIntersection* and *lodXMultiSurface* with $(X = 1, 2, 3, 4)$;
- *lodXMultiCurve* with $(X = 2, 3, 4)$;
- *lod0RoofEdge* and *boundedBySurface*.

4.5 Structural Changes

Changes reflecting structural modifications of city objects in the real world (such as a new constructed building part or a wall surface removed from an existing building) are considered as "structural changes", these include the following elements:

- *BuildingPart*;
- The boundary surfaces such as *RoofSurface*, *WallSurface*, *GroundSurface*, etc.

4.6 Top-level Changes

This last category covers changes that occur on the scale of top-level features, such as:

- Deletion of existing top-level features (such as buildings);
- Insertion of new top-level features (such as buildings).

Only the target node type *TType* and the child node type *CType* of the *InsertNode* and *DeleteNode* edit operations are required. These node types are then compared with the class name of the top-level features (such as *Building*). Both the options using graph queries shown in Section 3.2 and exported tables in Section 3.3 can be employed to determine top-level changes.

5. RELEVANCE OF CHANGE CATEGORIES WITH RESPECT TO USERS

The different edit operations and categories defined in Section 3 and 4 give an overview of both the literal and semantic aspect of the detected changes. They provide the key information to help better analyse and understand how cities progress. This can however be interpreted differently depending on the different needs, requirements and expectations of users and stakeholders. This section introduces three groups of most relevant users and stakeholders as follows :

- City administrators and municipalities: These stakeholders are in charge of planning and important decision making. They are mostly interested in large-scale changes such as the number of buildings that have been inserted or deleted from the city models, the number of buildings that are unchanged, as well as which buildings have been recently renovated and received new walls and windows, etc.
- Data administrators and surveyors: These stakeholders manage and maintain virtual city models on a regular basis. Some of their concerns include e.g. the minimum number of which edit operations should be executed in order to keep the datasets up-to-date, as well as how to keep track of real changes, etc.
- Developers and programmers: These stakeholders are in charge of the technical development and implementation of new concepts and methods. They are interested in understanding, analysing and experimenting on changes found. This includes questions such as whether the coordinate reference system (CRS) in a dataset has been changed, how a change in software used to manage datasets can have an impact on all types of changes in the datasets, etc.

An example of the relevance and interest levels of different categories of changes with respect to different groups of users and stakeholders is illustrated in Table 2. Note that there are generally different methods, factors and criteria to consider while classifying the interests and concerns of users and stakeholders. The example grouping of users presented above as well as the relevance levels shown in Table 2 should therefore be considered as an initial attempt to provide a better overview of how users interact with different types of changes.

	City Administrators	Data Administrators	Developers/ Programmers
Procedural Changes	○○	○●	●●
Thematic Changes	○●	●●	●●
Syntactic Changes	○○	○○	●●
Geometric Changes	○●	●●	●●
Structural Changes	○●	●●	●●
Top-level Changes	●●	●●	●●

○○ not relevant ○● less relevant ●● very relevant

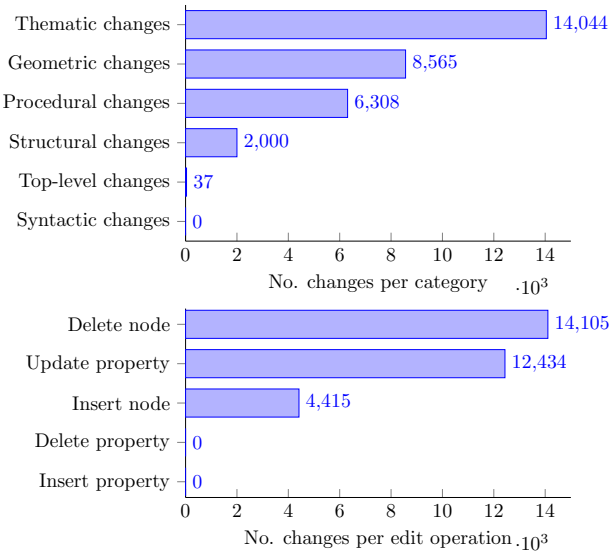
Table 2. An example of the relevance of different categories of changes with respect to different groups of users and stakeholders.

6. APPLICATION RESULTS

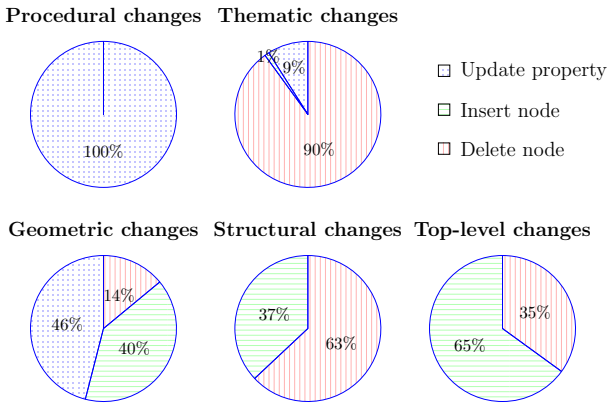
The proposed extended concepts of edit operations in Section 3 and different categories of changes in Section 4 shall be tested against two different use cases. In the first test case with smaller CityGML datasets from the district Moabit of the city of Berlin, the relations between the edit operations and categorized changes along with other findings shall be discussed. In the second test case with much larger CityGML datasets covering the entire German state of North Rhine-Westphalia, the scaling abilities of the proposed concepts shall be evaluated. Both the test cases were performed on a machine running SUSE Linux Enterprise Server 12 SP1 (64 bit) equipped with Intel® Xeon® CPU E5-2667 v3 at 3.20GHz (16 CPUs + Hyper-threading), a PCIe Solid-state Drive Array (SSD) and 1 TB of main memory.

6.1 Test Case Berlin Moabit

Two input CityGML datasets of the same area of the district Moabit of the city Berlin shall be tested. The older and newer CityGML dataset were recorded in 2008 and 2015, are 12 MB and 17 MB in size and contain 642 and 653 buildings respectively. Both datasets are given in LOD2.



(a) Number of detected changes per category and edit operation



(b) Distribution of edit operations per category

Figure 4. Applications results in the test case of Berlin Moabit.

A total number of 237,222 nodes and 30,954 edit operations were created in the graph database. The matching process took 93 seconds. The application results summarized in Figure 4 show that 80% (24,646) of all detected changes are considered as "real" changes. These include thematic, geometric, structural and top-level changes. Deleted thematic data account for 12,603 (41%) of all changes and are therefore the most common type of changes in this test case. Changes in generic attributes account for 99% of all thematic changes. In addition, no syntactic changes were found. This could be due to the fact that both the datasets were exported using the same software. Stakeholders such as city administrators can make use of the top-level changes to determine the number of buildings inserted to the newer city model (24) as well as the number of buildings deleted from the older city model (13).

Interestingly, as shown in Figure 5, a height difference of about 40 meters between all buildings of the older and newer dataset was found by manual inspection. Due to this significant difference in height, instead of using (3D) bounding boxes, (2D) footprints of buildings were used to determine the best matching candidates for each building. Such systematic changes are currently not categorized and captured by the implementation.

Figure 5. The systematic height difference of almost 40 meters between the older (upper) and newer city model of Berlin Moabit. Visualized by the 3DCityDB Web Map Client (Yao et al., 2018).

6.2 Test Case North Rhine-Westphalia

In the second test case, the scaling capabilities of the extended implementation shall be tested against very large amount of data. Two input city models in LOD1 recorded in 2016 and 2018 of the entire German state of North Rhine-Westphalia shall be tested. The older and newer city model are 79 GB and 89 GB in size and contain 10,132,245 and 10,459,646 buildings respectively. Since each of the dataset is provided in 35,022 tiles, the implementation was also extended accordingly to detect, map and match each pair of the tiled datasets of the same area in a multi-threaded manner. The generated edit operations for each pair of tiles were then totalled up. The results are shown in Table 3.

A total number of 1,798,612,970 nodes and 338,335,694 edit operations were created in the graph database (see Table 3) occupying 1.4 TB of disk space. The matching process took approximately 30 hours including the time needed to initialize a graph database instance and fill it with CityGML objects for each of the 35,022 tile pairs. The test results show that 61% of all detected changes are procedural, 94% of which are caused by modified identifiers. A total number of 123,879,652 or 37% of all changes are considered "real" changes, most of which are geometric. 294,877 buildings have been deleted from the older dataset and 622,678 have been inserted to the newer dataset.

	Insert Property	Delete Property	Update Property	Insert Node	Delete Node	
Procedural Changes	0	194,315,795	13,737,194	0	0	**208,052,989**
Thematic Changes	2,060,189	0	14,265,699	10,210,495	4,195,900	**30,732,283**
Syntactic Changes	0	0	6,351,626	3,465	47,962	**6,403,053**
Geometric Changes	0	0	85,567,419	1,198,427	3,043,357	**89,809,203**
Structural Changes	0	0	0	2,380,944	39,667	**2,420,611**
Top-level Changes	0	0	0	622,678	294,877	**917,555**
	2,060,189	**194,315,795**	**119,921,938**	**14,416,009**	**7,621,763**	**338,335,694**

Table 3. Number of detected changes per edit operation (columns) and category (rows) in the test case of North Rhine-Westphalia. The last column and row contain the total number of all cells of the respective rows and columns.

7. CONCLUSION AND FUTURE WORK

This paper extended the conceptual model and definition of edit operations to enable efficient querying and analysing on real detected changes. In addition, a new concept was proposed to further divide edit operations in categories based on their semantic contents, namely procedural, thematic, syntactic, geometric, structural and top-level changes. This set of different types of categories and edit operations enables a multi-perspective approach to interpreting detected changes in a more user-friendly and oriented manner. The concepts and implementation[1] presented in this paper are initial attempts at providing a better overview of how users could interact with different types of edit operations and categories of changes. This should allow for more quantitative testing in this field and shall be further discussed in the future.

The current implementation is capable of detecting and classifying changes efficiently in large CityGML datasets. However, as observed in Section 6.2, the tool slows down when it is applied to a large number of small input tile datasets. This could be improved by, given enough hardware resources, mapping all tiles onto a single graph database before executing the matching process. Additionally, the implementation could be customized for a specific group of stakeholders by only targeting relevant changes.

On the other hand, systematic changes, such as an elevation in height of all buildings observed in Section 6.1, could not be detected automatically in the current implementation. This is due to the fact that in which form such systematic changes may occur in the datasets and to which extent these may have an impact on other elements (e.g. a systematic height elevation of all buildings also results in a series of changed height coordinates of all geometric elements of those buildings) are not yet fully studied. Therefore, the current methods and implementation could be extended to detect such systematic changes automatically in the future.

ACKNOWLEDGEMENTS

We acknowledge the company CADFEM and the Leonhard Obermeyer Center (LOC) of the Technical University of Munich (TUM) for supporting this work. We also would like to thank the Bavarian Agency for Digitisation, High-Speed Internet and Surveying (LDBV) and the state government of North Rhine-Westphalia for providing the input datasets. We would like to thank Ordnance Survey GB (*www.ordnancesurvey.co.uk*) and 1Spatial (*www.1spatial.com*) for sponsoring the publication of this paper.

[1]The software developed in this research is open source and available under *https://github.com/tum-gis/citygml-change-detection*.

References

[1] A. Agoub, F. Kunde, and M. Kada. “Potential of graph databases in representing and enriching standardized Geodata.” In: *Tagungsband der* 36 (2016), pp. 208–216.

[2] M. Bakillah, Y. Bédard, M. A. Mostafavi, and J. Brodeur. “SIM-NET: A View-Based Semantic Similarity Model for Ad Hoc Networks of Geospatial Databases.” In: *T. GIS* 13.5-6 (2009).

[3] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. “Change Detection in Hierarchically Structured Information.” In: *SIGMOD Rec.* 25.2 (June 1996).

[4] K. Falkowski and J. Ebert. “Graph-based urban object model processing.” In: *City Models, Roads and Traffic (CMRT’09): Object Extraction for 3D City Models, Road Databases and Traffic Monitoring-Concepts, Algorithms and Evaluation, Paris, France* 9 (2009).

[5] G. Gröger, T. H. Kolbe, C. Nagel, and K.-H. Häfele. *OpenGIS(R) City Geography Markup Language (CityGML) Encoding Standard*. Version: 2.0.0. OGC. Apr. 4, 2012.

[6] J. W. Hunt and T. G. Szymanski. “A Fast Algorithm for Computing Longest Common Subsequences.” In: *Commun. ACM* 20.5 (May 1977), pp. 350–353.

[7] E. W. Myers. “AnO (ND) difference algorithm and its variations.” In: *Algorithmica* 1.1-4 (1986), pp. 251–266.

[8] S. H. Nguyen, Z. Yao, and T. H. Kolbe. “Spatio-Semantic Comparison of Large 3D City Models in CityGML Using a Graph Database.” en. In: *Proceedings of the 12th International 3D GeoInfo Conference 2017*. Vol. IV-4/W5. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. University of Melbourne. Melbourne, Australia: ISPRS, 2017, pp. 99–106.

[9] R. Redweik and T. Becker. “Change Detection in CityGML Documents.” In: *3D Geoinformation Science: The Selected Papers of the 3D GeoInfo 2014*. Springer, 2015.

[10] Y. Wang, D. J. DeWitt, and J. Y. Cai. “X-Diff: an effective change detection algorithm for XML documents.” In: *Data Engineering, 2003. Proceedings. 19th International Conference*. Mar. 2003.

[11] Z. Yao, C. Nagel, F. Kunde, G. Hudra, P. Willkomm, A. Donaubauer, T. Adolphi, and T. H. Kolbe. “3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML.” en. In: *Open Geospatial Data, Software and Standards* 3.5 (2018), pp. 1–26.