Potential of Graph Databases in Representing and Enriching Standardized Geodata

AMGAD AGOUB[1], FELIX KUNDE[2] & MARTIN KADA[1]

Abstract: The storage of spatial data in a database usually happens in relational database management system (RDBMS) such as PostgreSQL/PostGIS or Oracle Spatial. Due to the layer-based thematic modeling used in present-day Geographical Information Systems (GIS), the database schemas most often comprise a rather flat structure. However, the trend in spatial models, e.g. by the Open Geospatial Consortium (OGC), in international standards goes for more complex application schemas with well-defined objects, attributes, and relations. Mapping these (most often UML-based) data models into a RDBMS is challenging. Different solutions exist to guarantee a generic mapping, providing a more or less useful database schema to work with. But even these tools can fall short when dynamically supporting multiple user-defined extensions that some OGC standards have started to offer (e.g. CityGML).
Therefore, a growing interest for modern NoSQL systems can be noted. They are, by nature, schema-less and often very well suited for web-based applications - albeit their limited spatial functionality, data migrating and testing is often easier as with RDBMS. In particular, graph databases provide a structure that could help dealing with this problem as the underlying data model of nodes and edges can natively represent a conceptual UML diagram. This paper will present a lightweight mapping approach that supports on-the-fly mapping and storage for various OGC standards (i.e. SensorML, CityGML, etc.) into Neo4j, a pure graph-oriented database, and ArangoDB, a recently-developed multi-model database. The chosen DBMS are evaluated in terms of usability and support for spatial data management.

1 Introduction

The Open Geospatial Consortium (OGC) has developed many implementation standards (Simple Features, Open Web Services, Geography Markup Language) that are now widely adopted in commercial and open source software. This has helped to design, plan, and build interoperable spatial data infrastructures all over the world. Within the last 10 years new standards were introduced focusing on domain-specific location-based data models, ranging from sensor properties (SensorML) to urban modeling (CityGML)[3]. The data models of the Infrastructure for Spatial Information in the European Community (INSPIRE) are also greatly based on the work by the OGC. Therefore, the ability to read, store and write data following these application schemas will be a key to the success of such initiatives.

[1]Technische Universität Berlin, Institut für Geodäsie und Geoinformationstechnik, Straße des 17. Juni 135, D-10623 Berlin, E-Mail: amgadagoub@gmail.com, martin.kada@tu-berlin.de
[2]Beuth Hochschule Berlin, Labor Rechner- und Informationssysteme, Luxemburger Straße 10, D-13353 Berlin, E-Mail: fkunde@beuth-hochschule.de
[3]http://www.ogcnetwork.net/gmlprofiles

2 Limitations for mapping into relational databases

Different solutions have been developed to store and manage standardized data models inside a spatial database. In most cases relational database management systems (RDBMS) are used because of their maturity, their wide acceptance and their capabilities in processing geodata (e.g. Oracle Spatial, PostGIS, SpatiaLite or SQL Server).

Mapping object-oriented OGC data models into compact relational schemas without losing information is a challenging task ("impedance mismatch"). For a generic automatic mapping the class graph of a UML model can be used to define a set of algorithms and transformation rules to map the syntactical structure into an entity-relationship model (ER) (RODRIGUEZ et al. 2011). This can be done on behalf of the XML Schema Definition (XSD) and/or the XML Metadata Interchange (XMI). Such an approach could lead to a relatively complex relational model with a high number of tables (e.g. deegree[4], Go Loader[5]) or a very generic meta-model (e.g. SupportGIS[6]). Thus, writing queries could require using many (self-)joins either ways which has an effect on the performance. As a consequence, the deegree project provides an additional BLOB-Mode for fast data retrieval.

Finding a good balance between schema design and usability also requires a deep knowledge about the target domain which makes an automatic mapping even harder. The 3D City Database project[7] applies a semi-automatic mapping, where the UML class diagram of the OGC standard CityGML is first simplified manually with respect to performance and later use cases before converting it into an entity relationship model. Despite the advantage of having a more query-friendly database schema, an additional manual mapping step reduces the flexibility in serving arbitrary OGC schemas. The CityGML standard itself allows for extending and creating feature classes via the Application Domain Extension (ADE) mechanism (VAN DEN BRINK et al. 2014), which is not yet supported by the 3D City Database at the moment.

3 NoSQL databases as an alternative?

NoSQL ("not only SQL") has become a synonym for a big family of either new database systems that have been developed to fulfill modern business requirements of the internet age or even for older DBMSs that do not store their data in tabular relations. A common characteristic by many NoSQL systems is a high flexibility for storing very heterogeneous data. This feature raises the question if using a NoSQL database might overcome the limitations described in chapter 2.

3.1 Document-oriented databases

Data that is described using an OGC standard is written to an instance document of the UML information model which is encoded in XML. These documents can directly be imported in a document-oriented database that is able to store any kind of documents in the same encoding. KOCH (2015) evaluated BaseX, a native XML database, as a possible candidate for storing

[4]http://download.deegree.org/documentation/3.3.13/html/featurestores.html
[5]http://www.snowflakesoftware.com/products/goloader/
[6]http://www.cpa-software.de/index.php?do=bas&do2=db&lang=d (in german)
[7]http://www.3dcitydb.net

CityGML data. BaseX provides a flexible schema control and is very fast on imports and exports. However, performance of analytical spatial queries has been considerably slower compared to an RDBMS (PostGIS).
MAO et al. (2014) created a 3D city model management that can scale well on cluster-based architectures. Therefore, they used MongoDB, a popular document-oriented database that manages data in collections of binary JSON documents. The conversion (unmarshalling) of CityGML files is done through a java binding library[8]. The database is comprised of three collections in order to serve three different kinds of queries (metadata, thematic and spatial features, 3D for visualization).

3.2 Graph databases

The underlying data model of graph databases follows the property-graph model consisting of nodes and edges which can natively represent a conceptual UML diagram[9]. FALKOWSKI & EBERT (2009) discussed a graph-based schema for integrated models of urban data which are represented as CityGML. In his opinion explicit graph representations can be used for processing semantics, geometry, topology and appearance of the city model. DELFOSSE et al. (2012) presented two methods of mapping a UML class diagram to a graph model. The first approach suggests an implicit mapping, where the data is stored without semantic schema information. The second approach includes the semantics into the graph data model as additional nodes which are connected to each instance of the corresponding feature class ("Instance of relationship").

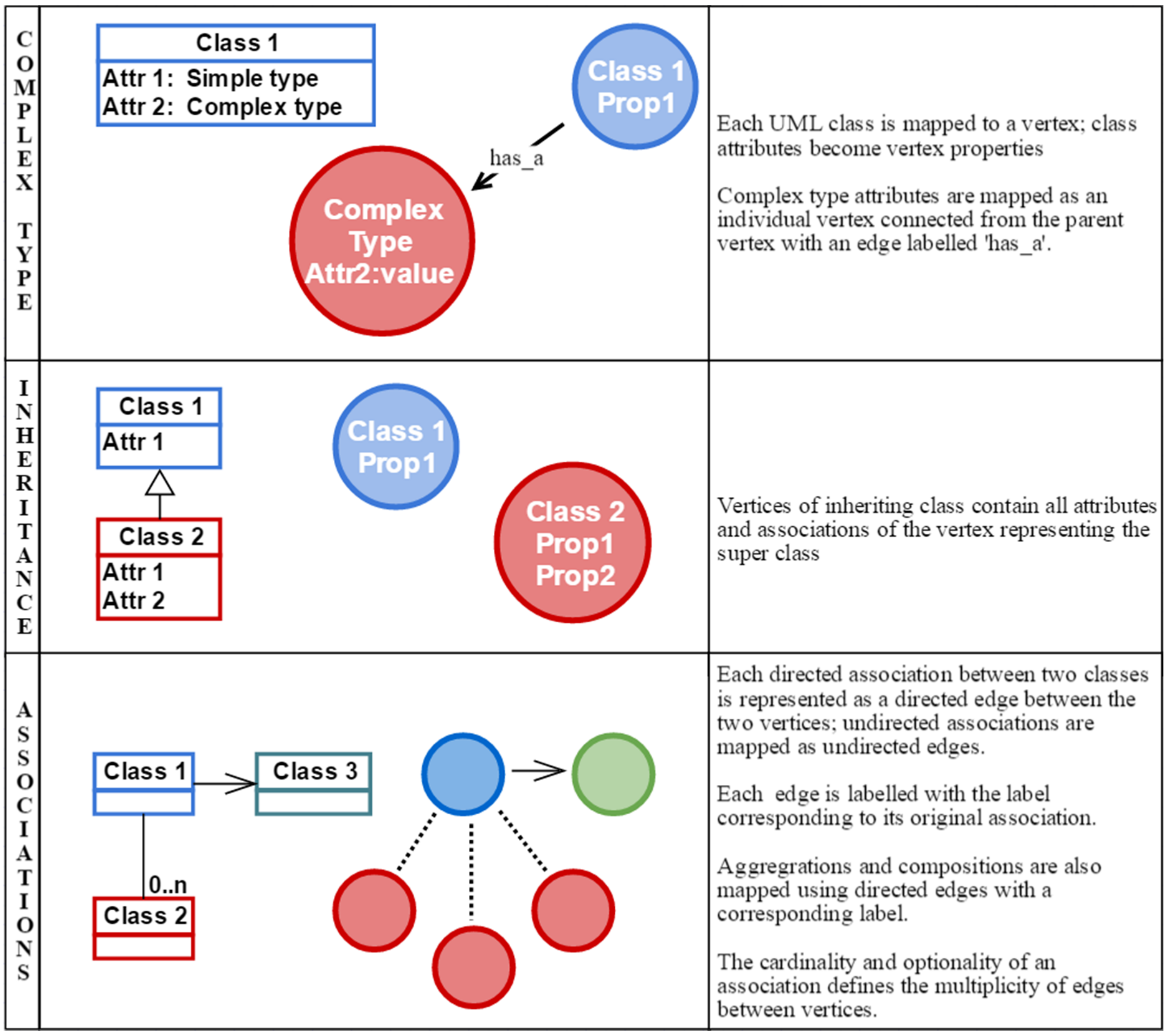
3.3 Polyglot persistence and multi-model databases

Each type of NoSQL databases is suitable for specific data and use cases. Sometimes it can be necessary to use more than one database to encompass all user requirements. This strategy is called 'polyglot persistence'. BOYD (2015) presented a back end coupling MongoDB for data storage with Neo4j for shortest path queries[10]. A similar concept is thinkable for OGC standards where objects would form semantically classified collections of documents and their relationships (linkage, recursion, versioning etc.) would be expressed through graphs.
Orchestrating different database servers often creates an overhead in maintenance, software development and data throughput. Multi-model databases combine different storage models in one software to overcome these issues. Two modern DBMSs combining the document and the graph model are OrientDB and ArangoDB. However, both systems are still lacking a sophisticated spatial support. In 2013, a GraphGIS extension was introduced for OrientDB (HOUBIE 2013), but the project has been inactive since then.

4 Mapping concept

This section proposes a concept of mapping the schema definition of OGC standards to a property-graph model in order to store OGC-compliant instance documents into a graph database. The applied transformation rules are based on the theoretical work of DELFOSSE et al. (2012) with some minor changes. Fig. 1 provides a general overview of how a UML class diagram is mapped:

[8]https://github.com/citygml4j/citygml4j
[9]http://lambdazen.blogspot.de/2014/01/from-entity-relationship-to-property.html
[10]http://neo4j.com/blog/polyglot-persistence-mongodb-wanderu-case-study/



Each UML class is mapped to a vertex; class attributes become vertex properties

Complex type attributes are mapped as an individual vertex connected from the parent vertex with an edge labelled 'has_a'.

Vertices of inheriting class contain all attributes and associations of the vertex representing the super class

Each directed association between two classes is represented as a directed edge between the two vertices; undirected associations are mapped as undirected edges.

Each edge is labelled with the label corresponding to its original association.

Aggregrations and compositions are also mapped using directed edges with a corresponding label.

The cardinality and optionality of an association defines the multiplicity of edges between vertices.

Fig. 1: Overview of the mapping context.

4.1 Generic import of OGC XML documents

A machine readable class graph of the UML model can be parsed from an XSD file. Fortunately, most of the OGC standards are already mapped to XSD files by the OGC itself and can be downloaded from its official home page: http://schemas.opengis.net/ (2016). The download process can be automated while parsing an XML file if it contains a reference to the schema location in its root element[11].

An import process begins with binding the elements of an XSD schema file to a set of classes that represent the schema in the programming language used for the importer software. These classes define the mapping rules (see Fig. 1) which are further applied for unmarshalling the XML data into objects of the used binding. The mapping of XML data types should be type-safe to guarantee consistency within the target database.

[11]https://www.w3.org/TR/xmlschema-1/

A challenge when mapping from an XSD file is to differentiate between features, relationships and non-primitive attributes in the later graph model because all of them are encoded as complex types (xs:complexType) and create the same structure of edges and vertices based on the transformation rules. For this reason, associations are mapped to connecting vertices between two features instead of having only an edge. For imports into a multi-model database this step requires additional software logic to create a working structure.

4.2 Writing bind objects to graph data models

The objects from the XML binding can be prepared for imports into either a pure graph database or hybrid graph DB that has the ability to use other data storage models in conjunction (see multi-model databases in chapter 3). A unique identifier is generated and added to each vertex as a property. For depending child nodes this identifier is used to add a reference property. It is a common strategy for graph databases to collect many vertices in bigger batches to reduce processing time and network traffic. The edges are written afterwards on behalf of the reference properties.

4.3 Importer architecture

The introduced mapping and import concept has been realized as a client-server application and is written in JavaScript. Although powerful XML binding frameworks exist for languages like Java (e.g. JAXB – Java Architecture for XML Binding), JavaScript had been chosen to create a lightweight prototype that is easy to deploy on every operating system and machine or device. The server application is built with Node.js, a runtime environment built on Google's V8 JavaScript engine. Node.js has a very active community that provides many different kinds of plugins and application programming interfaces (API) including also drivers for graph databases.
The importer core depends mainly on two Node.js packages: Jsonix[12] and OGC Schemas[13]. Jsonix is used to unmarshal XML into JSON objects (JavaScript Object Notation) and can be seen as the JavaScript equivalent to JAXB. OGC Schemas provides JAXB and Jsonix bindings for all OGC standards. However, it is not an official OGC project. The classes of these bindings provide the context for converting the schema elements of the XML file in corresponding JSON objects. This conversion step is type-safe because it depends on the file's schema and not a generic JSON parser.
Two databases were chosen to test the mapping approaches explained in this chapter. As for a pure graph DBMS Neoj4[14] was selected, because it offers a spatial plugin which would be a must-have feature for being a real alternative to relational geodatabase systems. For a multi-modal database the rather young project ArangoDB[15] was tested, because it offers an additional JavaScript layer (Foxx) for creating additional software logic to keep the database schema consistent and OGC-compliant. Foxx integrates well into the proposed architecture.

[12]https://github.com/highsource/jsonix
[13]https://github.com/highsource/ogc-schemas
[14]http://neo4j.com/
[15]https://www.arangodb.com/

5 Use cases and results

5.1 Mapping to Neo4j

Vertices were imported into Neo4j and then edges were created with a query written in Cypher. Cypher has been developed by the Neo4j developers and is the default query language of Neo4j. It is declarative by nature and very similar to SQL. Queries to retrieve features were comparably shorter than their equivalents in RDBMS. The following code is an SQL query to retrieve the geometric and appearance properties of a single building from the 3D City Database database schema.

```
SELECT sg.*, tp.*, sd.*, a.* FROM building b
JOIN thematic_surface ts ON ts.building_id = b.id
JOIN surface_geometry sg ON sg.root_id = ts.lod2_multi_surface_id
LEFT JOIN textureparam tp ON tp.surface_geometry_id = sg.id
LEFT JOIN surface_data sd ON sd.id = tp.surface_data_id
LEFT JOIN appear_to_surface_data ats ON ats.surface_data_id = sd.id
LEFT JOIN appearance a ON a.id = ats.appearance_id
WHERE b.id = ?
```

The equivalent query in Cypher is as follows and yields the following result as shown in Figure 2.

```
Match (B:Building {B.id:id})-[:REL*]->(m)
return B,m
```

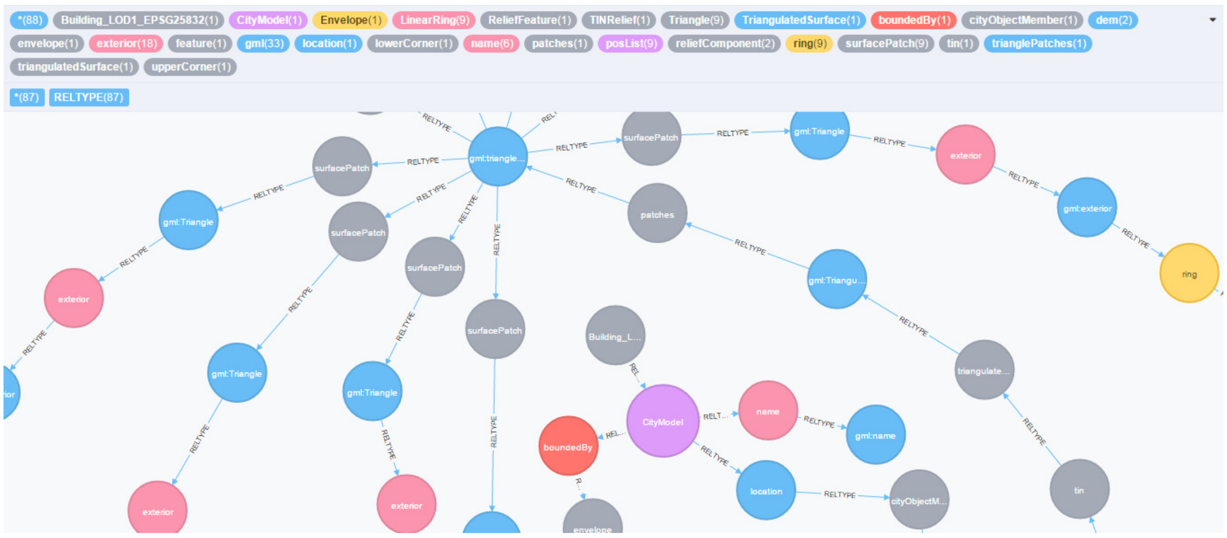


Fig. 2: CityGML dataset of a 3D building model ('Schloss Charlottenburg') in LOD 2 stored in Neo4j.

Some limitations were noted when using this mapping approach. Neo4j currently does not have support for GML geometry types out of the box. Therefore, GML data is stored in long chains of nodes as shown in Figure 3. If a GML-compliant geometry type would be integrated in Neo4j the resulting structure would be reduced to a single node or even to a node property with type GML.

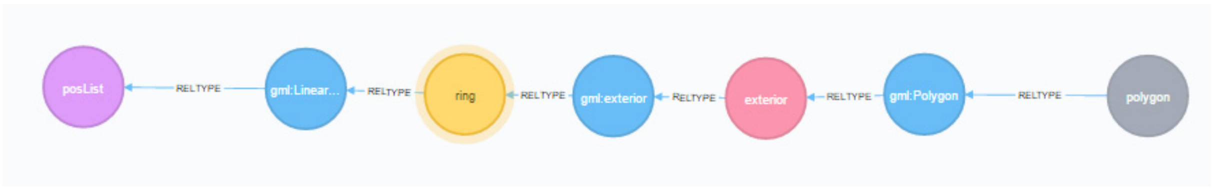


Fig. 3: GML polygon represented as a chain of nodes in Neo4j database which corresponds to its original XML tree structure.

Neo4j offers a native support for a variety of spatial capabilities and can be extended further with the 'Neo4j-spatial' extension[16]. Neo4j-spatial provides the possibility to store geometries as a node property with the type WKB (Well Known Binary). However, they are currently limited to 2D representation and the only supported coordinate reference system (CRS) is WGS84 (World Geodetic System 1984) (BAAS 2012). This would not be sufficient for many domain-specific data models of the OGC, e.g. CityGML, which targets mostly 3D use cases. Coordinate transformations would always be mandatory if the CRS is not WGS84, which generates an overhead for imports and exports.
Another limitation refers to the object hierarchies of the XML document tree which can only be mapped to parent-child relationships in Neo4j. This does not gather the full potential of graph databases because relationships are not fully exploited in this case. Further development into this direction can leverage the usage of relationships in the semantic context.

5.2 Mapping to ArangoDB

ArangoDB can natively store a nested JSON object as a data entry inside a collection. Therefore, there is no need to disassemble the resulting JSON objects. Thus, the stored data would simply inherit the tree structure of the XML data. This makes indexing and querying considerably harder in contrast to simple scalar (flat) documents.
The schema-free nature of ArangoDB enables the storage of data of arbitrary schema definitions and attributes without validation. But the database can be extended seamlessly through the Foxx framework mentioned in chapter 4.3. It can be used to create a schema validation for each added schema which can be applied through a http request. This layer was tested on a basic JSON schema as a proof of concept. Test data could be validated successfully against the schema. The proposed architecture for mapping and importing OGC documents on-the-fly to ArangoDB is shown in Figure 4. The unmarshalling to JSON would be similar to the Neo4j example.

[16]https://github.com/neo4j-contrib/spatial

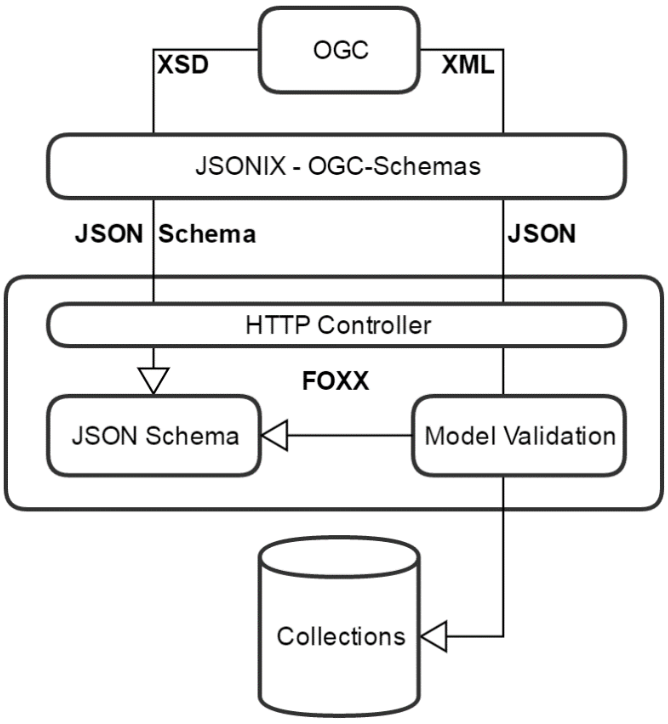


Fig. 4: Proposed mapping approach for importing OGC data into ArangoDB.

Several advantages should be noted when using the multi-model approach. First of all, each feature is mapped as an individual entry (document) in the database. Therefore, it is possible to retrieve the feature by calling its unique identifier yielding the following query as an equivalent for the aforementioned SQL query:

```
For x in Building
Filter x.id = = id
Return x
```

Secondly, in many cases real world objects can be represented in multiple files formats. For example, a city model can be described using CityGML, but for web visualization formats like glTF or COLLADA are better suitable. It would not make much sense to store these files in the database but their metadata can be linked to the corresponding documents for the OGC base data. Currently, ArangoDB has more spatial limitations than Neo4j. It supports only points as a geometric representation, only one reference system (WGS84) and a few spatial query operations[17].

6 Conclusion

In this paper, we presented a prototype that should prove the suitability of graph databases for storing and managing OGC standard documents. The schema-less design facilitates to store data from different domain-specific data models and any user-defined application extensions to these

[17]https://docs.arangodb.com/SimpleQueries/GeoQueries.html

models in one database. However, it is still possible to create a validation layer in between to guarantee standard compliance and data consistency. Especially, multi-model databases make it easy to create additional conformance requirements that data imports need to pass.
The spatial support for the tested DBMS (and many NoSQL databases in general) still falls short compared to object-relational databases. At the moment they might only be used for storage and simple thematic and spatial queries rather than advanced location intelligence (LI) tasks. But with a worldwide growing GIS market and GIS community there is enough potential to generate demands to extend these software products. An easy-to-use and easy-to-deploy data mapping and management solution like the developed example also helps to lower the boundaries for working with OGC standard data. The work presented some query examples that where much simpler than their relational counterparts.
Future work might focus on improving the mapping rules for multi-model databases. Exports from the graph model to OGC XML documents is not covered. At the moment the JavaScript code is not designed to consume massive data sets, as this research is a proof of concept.

7 References

BAAS, B., 2012: NoSQL spatial: Neo4j versus PosGIS. Master thesis, Delft University of Technology.

BOYD, R., 2015: Neo4j Blog - Polyglot Persistence Case Study, 2015: Wanderu + Neo4j + MongoDB - Neo4j Graph Database. Neo4j Graph Database. Available at: http://neo4j.com/blog/polyglot-persistence-mongodb-wanderu-case-study/ [Accessed 28 Apr. 2016].

DELFOSSE, V., BILLEN, R. & LECLERCQ, P., 2012: UML as a schema candidate for graph databases. NoSQL Matters.

FALKOWSKI, K. & EBERT, J., 2009: Graph-based urban object model processing. 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.

HOUBIE, F., 2013: GraphGIS, bringing spatial functionalities in NoSQL graph databases. FOSS4G – OSGeo's Global Conference for Open Source Geospatial Software, Nottingham, UK, 17-21 September.

KOCH, S., 2015: Storage and Querying of CityGML Models in BaseX - Evaluation of the Usage of a Native XML Database System for 3D City Models. Master Thesis, Technische Universität Berlin.

MAO, B., HARRIE, L., CAO, J., WU, Z. & SHEN, J., 2014: NoSQL Based 3D City Model Management System. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences **40** (4), 169.

RODRÍGUEZ, A., FERNÁNDEZ-MEDINA, E., TRUJILLO, J. & PIATTINI, M., 2011: Secure business process model specification through a UML 2.0 activity diagram profile. Decision Support Systems **51** (3), 446-465.

VAN DEN BRINK, L., STOTER, J. & ZLATANOVA, S., 2013: UML-Based Approach to Developing a CityGML Application Domain Extension. Transactions in GIS **17** (6), 920-942.