GRAPH-BASED URBAN OBJECT MODEL PROCESSING

Kerstin Falkowski and Jürgen Ebert

Institute for Software Technology
University of Koblenz-Landau
Universitätstr. 1, 56070 Koblenz, Germany
{falke|ebert}@uni-koblenz.de
http://www.uni-koblenz-landau.de/koblenz/fb4/institute/IST/AGEbert

KEY WORDS: Urban, Model, Metadata, Data Structures, Algorithms, Processing, Services.

ABSTRACT:

Urban object models are valuable assets that allow reuse in different applications. Besides the need for exchange formats there is also the need for comprehensive, efficiently processable data structures for such models. This paper presents a graph-based schema for integrated models of urban data, that is an adaption of the comprehensive CityGML approach. It defines an explicit graph representation and thus is well-suited to efficient processing algorithms. The paper demonstrates how appropriate light-weight components realizing different kinds of services on models can be used for consistently processing semantics, geometry, topology and/or appearance of graph-based models compliant to that schema. Several examples are given.

1 INTRODUCTION

Urban models are valuable assets that should be constructed once while being used multiple times in different applications. Therefore the exchange of 3d city models between different tools is indispensable. Various XML formats are being used to achieve interoperability between tools. These formats (e.g. CityGML (Groeger et al., 2008)) are able to carry topological, geometric, semantic, and appearance information, but in different forms and to varying extent.

Applications, like tools for the automatic extraction of topographic objects, build on these urban object models and improve, transform, and analyze them in different ways. XML-technology (e.g. XSLT and XQuery) is widely used to support these activities, but this technology is not well-suited for the implementation of the various algorithms on urban objects which come from the areas of algorithmic geometry, computer graphics, and image recognition, since the necessities of efficient content-based traversal of all relevant information is only hard to realize in the essentially tree-like structures supplied by XML.

Therefore, a comprehensive, efficiently processable data structure for urban objects is essential. Geographic information systems share this necessity with route guidance systems, where a graph-like internal representation of data is used for the computation of routing information.

In this paper, we present an approach for the efficient storage, analysis, and manipulation of city models using graphs and for the development of application specific components collectively working on an integrated, efficient graph representation of city models. Import/export from/to CityGML is tackled, as well. [1]

After a short overview of the state of the art in section 1.1, section 2 shortly introduces the employed graph and component concepts. Section 3 describes the graph-based integrated model schema with all its aspects, and section 4 shows how quite different kinds of functionalities can be implemented on such a model by independent components. Section 5 concludes the paper.

[1] The project is funded by the DFG (EB 119/3-1).

1.1 State of the art

There are several XML-based modeling languages for urban objects. The *City Geography Markup Language (CityGML)*[2] is a common information model for the representation of 3d urban objects and an official standard of the Open Geospatial Consortium (OGC) since August 2008 (Groeger et al., 2008). Besides representing geometry, CityGML can also be used to model topological and semantic properties of 3d city models and to attach appearance information like textures.

Models described using CityGML can be rendered by *Ifc-Explorer for CityGML*[3] from the Institute for Applied Computer Science, Forschungszentrum Karlsruhe or the *LandXplorer CityGML Viewer*[4] from Autodesk and by *Aristoteles*[5] from the Institute for Cartography and Geoinformation, University of Bonn.

Besides CityGML there are other languages for the representation of 3d urban objects. One common approach is the OGC standard *Keyhole Markup Language (KML)*[6]. CityGML uses a subset of the OGC standard *Geography Markup Language (GML)* (Cox et al., 2001) for geometry representation, KML derived his geometric elements from GML. KML is often combined with the COLLADA[7] exchange format for 3d assets. Another 3d modeling language is *Extensible 3D (X3D)*[8], the successor of the *Virtual Reality Modeling Language (VRML)* standard.

2 BASIC TECHNOLOGIES

2.1 TGraph technology

For the efficient manipulation of urban object models with all their aspects a versatile and powerful basic technology is needed. In the context of this work TGraph technology is used.

[2] `http://www.citygml.org`, `http://www.citygmlwiki.org`
[3] `http://www.iai.fzk.de/www-extern/index.php?id=1570`
[4] `http://www.3dgeo.de/citygml.aspx`
[5] `http://www.ikg.uni-bonn.de/aristoteles`
[6] `http://www.opengeospatial.org/standards/kml`
[7] `http://www.khronos.org/collada`
[8] `http://www.web3d.org/x3d`

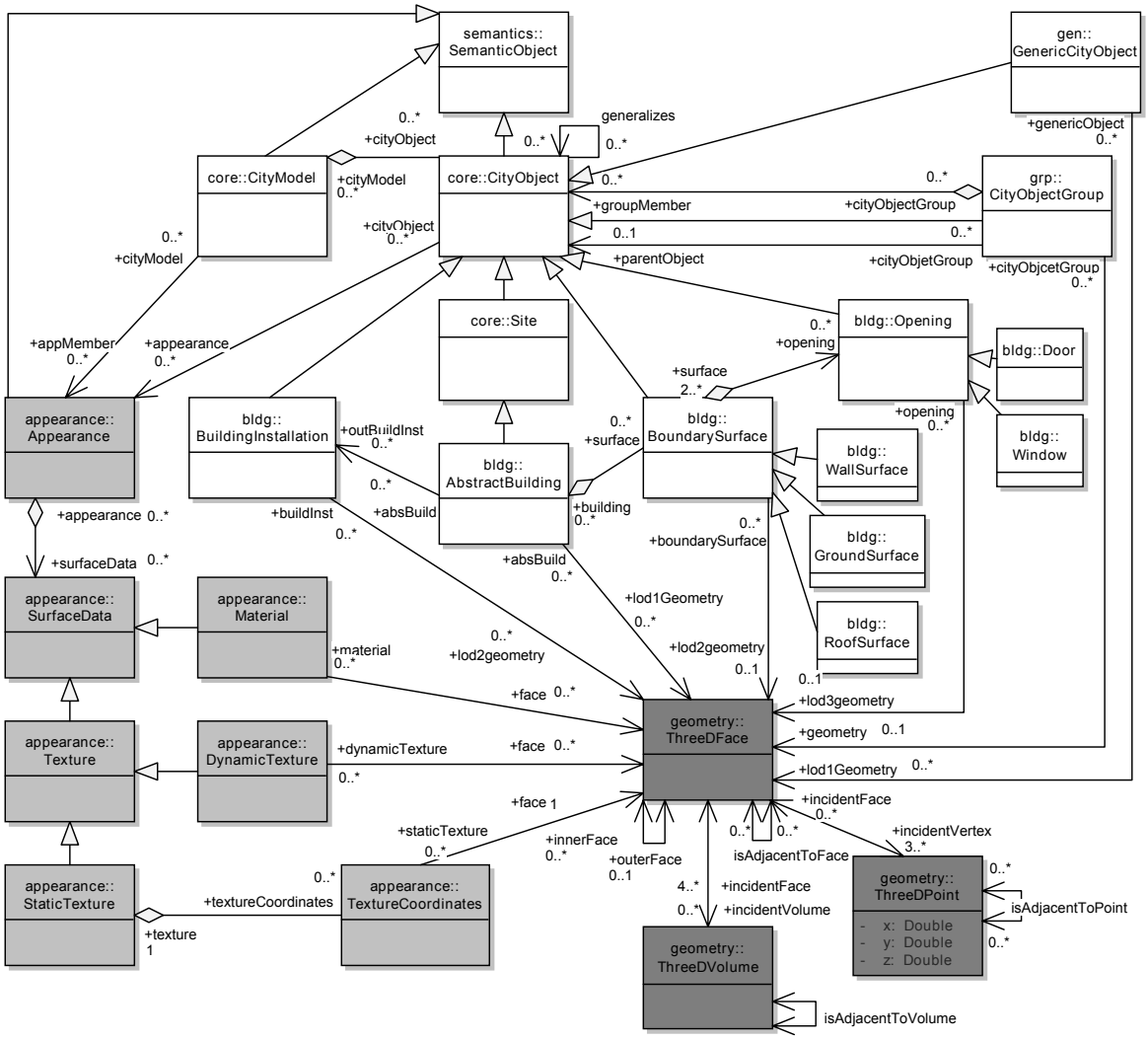

Figure 1: The integrated model schema as a grUML diagram. *Semantic entities* are colored white with namespace "sem", *appearance entities* are colored gray with namespace "app" and *geometry/topology entities* are colored dark gray with namespace "geo/top".

TGraphs are directed graphs whose vertices and edges are typed, ordered and attributed. Their structure, types and attributes help to model the different aspects (topology, geometry, semantics, and appearance annotation) of urban objects in a common integrated data structure. TGraphs are supported by a powerful API (JGraLab[9]) in combination with a *graph query language* (GReQL) and a corresponding UML-based *metamodeling approach* (grUML). grUML is a subset of UML class diagrams which allows the specification of classes of TGraphs on the schema level (Ebert et al., 2008). Figure 1 contains an example.

2.2 Lightweight component model

If all relevant data of the urban object model are stored in a TGraph, all processing of the model can be encapsulated in appropriate components working on this particular TGraph.

The work described here is based on a *light-weight Java component model* which is employed for the different processing activities on the model (see section 4). The component concept is basically an extension of the well-known *strategy pattern* (Gamma et al., 1995). Every component has a *definition* in the form of a Java interface which describes its service and at least one *implementation* in the form of a Java class.

[9] `http://jgralab.uni-koblenz.de`

Components are serializable and get their data to process as *arguments* of their `execute()`-methods. Further data that influence their work are handled as *parameters* which have a default value and are manipulated via *getters* and *setters*. For example some processing steps can be configured by parameters (like thresholds).

3 THE INTEGRATED MODEL SCHEMA

The internal representation of urban object models by TGraphs has to be specified by a metamodel, called *schema* in the following. This schema defines the set of compliant TGraphs. Classes define the possible vertex types, and associations define the edge types. The attributes of vertices and edges can be added according to the well-known UML notation, as well. Edge direction is visualized by arrow heads, though it should be noted, that TGraph edges are traversable in both directions by algorithms.

Figure 1 shows the main parts of an integrated schema which defines a set of TGraphs for urban objects. (To improve readability all enumeration types and some semantic subclasses as well as attributes are elided.) This schema is inspired by and partially derived from the CityGML 1.0 schema. Especially it follows the idea to separate the four relevant aspects of an urban object model (namely topology, geometry, semantics, and appearance).

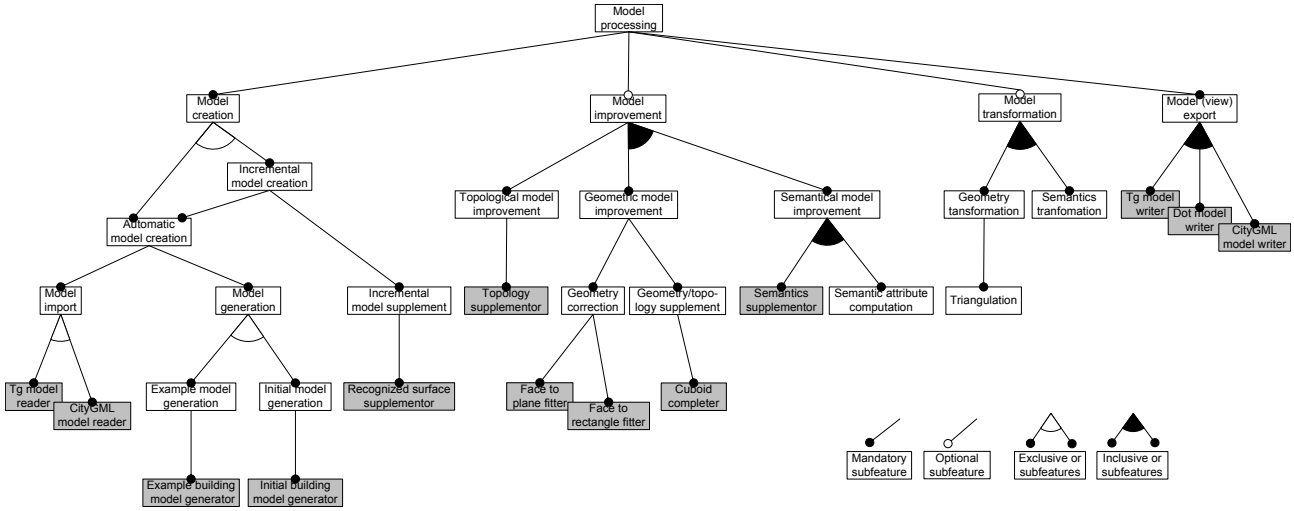

Figure 2: The main components as feature diagram. *Component groups* are colored white, *concrete components* are colored gray.

The schema contains *semantic entities*, *appearance entities* and *geometry/topology entities*. The semantic part contains entities from the subpackages Core (namespace "core"), CityObjectGroup (namespace "grp"), Generics (namespace "gen") and Building (namespace "bldg"). It should be noted that in principle also other ontologies might be used for the semantic part.

This grUML schema extends the tree-like XML schema of CityGML to a real graph-based schema, that (meta-)models the entities and relations of urban objects much more explicitly.

In CityGML models multiple occurences of the same object can mostly be modeled by defining the object once and referencing it using XLink[10]. But this is not possible for every object. As an example, the GML specification offers the definition of the control points of a `LinearRing` (exterior of a `Surface`) using the types `DirectPosition` or `PointProperty`. The first is used, if the control points are used only in this geometry element, the second is used used, if the control points may be referenced from other geometry elements. CityGML restricts these possibilities to `DirectPosition`. This means in CityGML models for every occurence of the same real world point as control point of the surfaces of a building there is a new `DirectPosition`. And even if XLinks are used, they often can not be processed sequentially and their interpretation is time-consuming.

Using the integrated model schema of figure 1, every entity in an urban object model exists only once as a node and all its uses and occurences are modeled by edges. This explicit, strongly linked representation reduces redundant information and enables automatic model processing by a very large class of algorithms. It is easy to import and export CityGML models using LODs 1-3 to and from an integrated model.

3.1 Geometry/topology schema part

The geometry/topology part of the integrated model schema differs from the CityGML schema. CityGML uses a subset of GML to represent geometric entities as a *boundary representation* (Foley et al., 1990, Herring, 2001). But the geometry/topology part of the integrated model schema is not based on this GML subset, since entities and relations are not represented explicitly enough and the same geometric objects may appear more than once in the same model.

[10]http://www.w3.org/TR/xlink

Here, geometry and topology are modeled as another kind of boundary representation, namely as an extended *vertex-edge-face-graph* (*v-e-f-graph*) similar to the well-known and highly efficient *Doubly Connected Edge List (DCEL)* representation (Muller and Preparata, 1978). A geometric object consists of *3d points*, *3d faces* and *3d volumes*, modeled as typed nodes connected via edges. The *geometric information* is encoded in the *attributes of the 3d points*, and the *topological information* is represented by the *edges between the geometric entities*.

3.2 Semantics schema part

The semantics part of the integrated model schema is based on the CityGML modules *Core*, *CityObjectGroup*, *Generics* and *Building*. Thus, terms like "building", "wall surface" and so forth can be used without further explanation in the following. Each of the mentioned modules is packed in its own subpackage.

3.3 Appearance schema part

The appearance part of the integrated model schema is oriented at the CityGML appearance module. But the different kinds of surface data (material, different kinds of textures) are directly related to the 3d faces they shall be applied to. The model allows *static and dynamic textures*, but dynamic textures are preferred. A dynamic texture consists of an image and a transformation matrix containing values to compute 2d texture coordinates for existing 3d points concerning the given image. By using dynamic textures, texture coordinates can be updated during model export if their corresponding 3d points have changed during model improvement.

4 INTEGRATED MODEL PROCESSING

The integrated model schema defines the class of TGraphs that represent urban object models with all their aspects. There are a lot of possible *processing activities* for integrated models, which are introduced in the following. Figure 2 gives an overview over such processing activities and their dependencies in the form of a *feature diagram* (Czarnecki and Eisenecker, 2000).

Here, the components constitute a product line (Pohl et al., 2005) where *features* are implemented by *Java components* (subsection 2.2). (The components are referenced by identifiers written in `typewriter` style.)

This chapter presents some of these processing components in more detail in order to prove on an example basis that all kinds of processing is possible on integrated models based on TGraphs.

The (intermediate) results of the different processing activities are exported using the `CityGMLModelWriter` component (subsection 4.5) and the XML text is rendered using the IfcExplorer for CityGML (section 1.1). IfcExplorer encodes different (semantic) parts of CityGML models using various colors. Wall surfaces are rendered gray, ground surfaces dark gray, roof surfaces red, doors dark blue, windows light blue and nearly transparent and all other faces cyan. (Unfortunately this distinction is hardly visible in the black-and-white versions of this article.)

4.1 Example

The functionality of the components is demonstrated on the basis of a model of one simple example building, which may be created using the `ExampleBuildingModelGenerator` (subsection 4.2). The full model consists of one ground surface, four wall surfaces, four roof surfaces, one door and five windows, its geometry contains fifteen 3d faces and thirty four 3d points. The user can choose, which model parts should be generated and how they should be connected. Figure 3 shows the full model. Since semantics, geometry and topology of this example model are well known, it is used as example model for most of the components mentioned in the following.

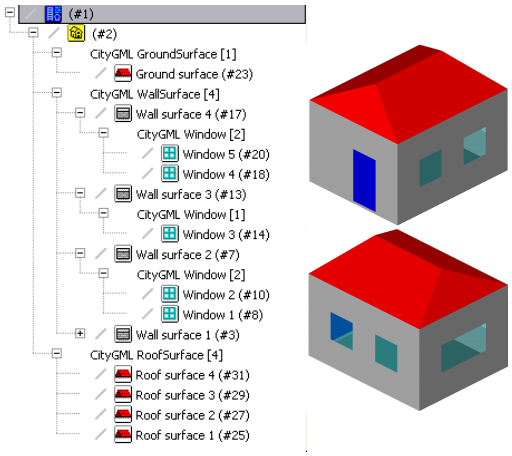

Figure 3: Full example model.

4.2 Model creation.

First of all, an integrated model has to be created. There are two kinds of automatic model creation, namely *model import* and *model generation*.

Model import. During model import an existing model is read from a file. `TgModelReader` reads an existing integrated model from a .tg-file (the JGraLab file format) and `CityGMLModelReader` reads an existing CityGML model from an .xml-file and transforms it into an integrated model. This import respects the CityGML[Appearance,Building,CityObjectGroup,Generics][11] profile. Semantic objects from other CityGML modules are ignored at present.

Model generation. During model generation an integrated model is created from scratch by a list of creation steps which are hard-coded in Java. The component `ExampleBuildingModelGenerator` constructs the complete example model from figure 3 that contains all four integrated model parts. `InitialBuildingModelGenerator` constructs incomplete models which function as bases for incremental model supplementation activities, which are not explained in further detail here.

[11] The Core module is not mentioned in CityGML profile names, because it belongs to every profile

4.3 Model improvement

The advantage of a graph-based representation of 3d models becomes clear if elaborate algorithmic activities are applied to them. Such activities are especially needed if the imported model is still unprecise and incomplete, for instance because it consists of raw data delivered by some object extraction tool (Falkowski et al., 2009).

Then, the raw models might have to be improved algorithmically. This includes *topological*, *geometric* and *semantic model improvement*. Geometric model improvement may even be specialized into *geometry correction* and *geometry/topology supplement*.

Topological model improvement. During topological model improvement different kinds of topological information are added to an integrated model. The component `TopologySupplementor` complements an integrated model by adding implicit topological dependencies as explicit arcs in the graph. It may connect all neighboring faces of a 3d face by `isAdjacentTo`-edges and all neighboring 3d points accordingly to a 3d point, if they are not related yet. Furthermore it may add all 3d faces that lie in another 3d face as inner faces. The component uses the geometric/topologic model part, but changes only topology.

Topological model improvement should always be the first improvement step, since most of the later processing steps are based on computational geometry algorithms that assume complete topological information.

Geometry correction. In raw models, the computed 3d coordinates are often only known approximately. This may lead to (slightly) distorted models. See figure 4 as an example.

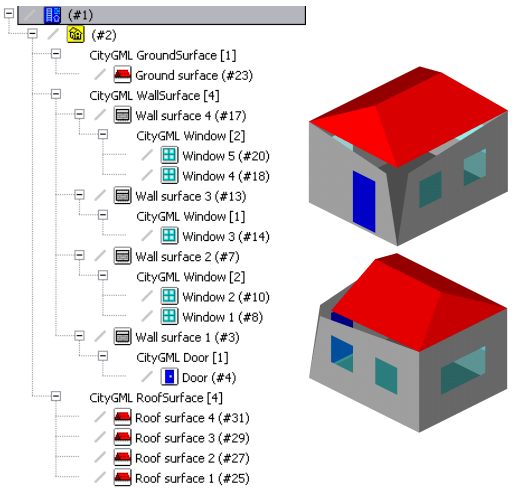

Figure 4: Geometry correction: Example model with "wrong" 3d point and therefore with 4 non-planar faces.

During geometry correction the geometry information of an integrated model (i.e. the x-, y- and z-coordinates of 3d points) is corrected. The `FaceToPlaneFitter` tests the planarity of all 3d faces and makes them planar, if they are not. The `FaceToRectangleFitter` tests the squareness of all 3d faces and makes

them rectangular, if they are nearly squared. Both components use appropriate approximation algorithms and both use the geometric/topologic model part, but change only the geometry. This correction transfers the model of figure 4 to the one in figure 3.

Geometry/topology supplement. Models extracted from 2d images are usually incomplete, since hidden information is missing. For urban data (sometimes) plausible assumptions may be made about the 3d-structure of the objects (e.g. they may be assumed to be cuboids).

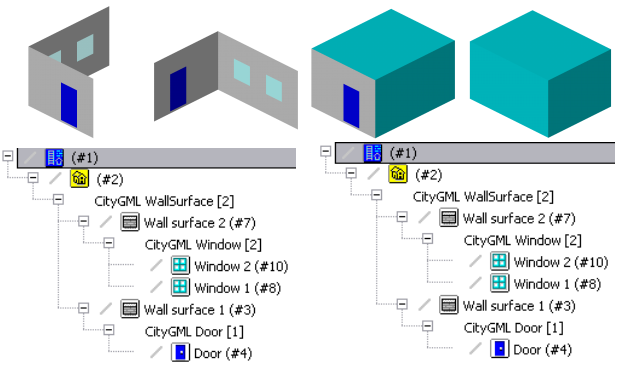

Figure 5: Geometry/topology supplement: Incomplete Example model (left), supplemented example model (right).

During geometry/topology supplement different kinds of geometric and topological information are added to an integrated model. The `CuboidCompleter` tests if there are incomplete cuboids in the integrated model and completes them by adding mirrored inverted copies of existing 3d faces (figure 5). The component uses the geometric/topologic model part, and enhances geometry as well as topology.

Semantic model improvement. Given a corrected and supplemented model, also semantic information might be inferable and should be added to the model.

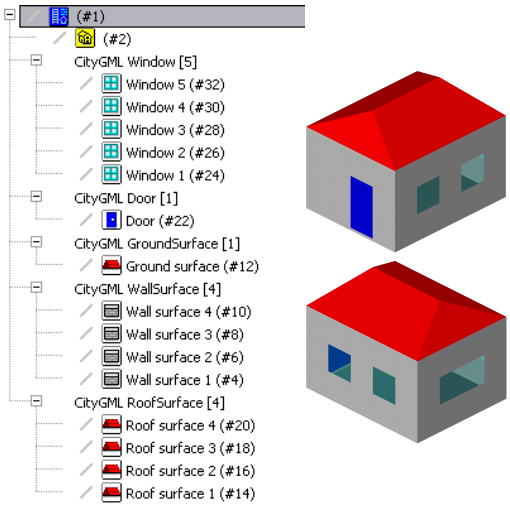

Figure 6: SemanticsSupplement: Model without relations between building and boundary surfaces as well as boundary surfaces and openings.

During semantic model improvement different kinds of semantic information are added to an integrated model. The component `SemanticsSupplementor` complements an integrated model by adding implicit semantic dependencies as explicit relations. It acts on the assumption, that if an object belongs to an aggregation, its parts also have to belong to this aggregation as well and vice versa. The component adds openings or building to a city model, if their related boundary surfaces belong to this city model. Figure 6 shows a an example of a semantically poor model which is be transformed into the full model of figure 3 by this component. The component uses the semantics and the geometric/topological model part, but changes only semantics.

4.4 Model transformation

A general class of processing activities is the modification of an integrated model by some kind of model transformation. There are *geometry/topology transformations* and *semantic transformations*. An example for geometry/topology transformation could be *triangulation*. An example for latter might be *changing the CityGML like semantics part* into one according to a proprietary ontology.

4.5 Model export

In general the integrated model or at least parts of it have to be stored persistently after processing. During model export an integrated model is written to a file.

The component `TgModelWriter` writes a full integrated model to a .tg-file. If the exported .tg-model is imported again, no information will be lost.

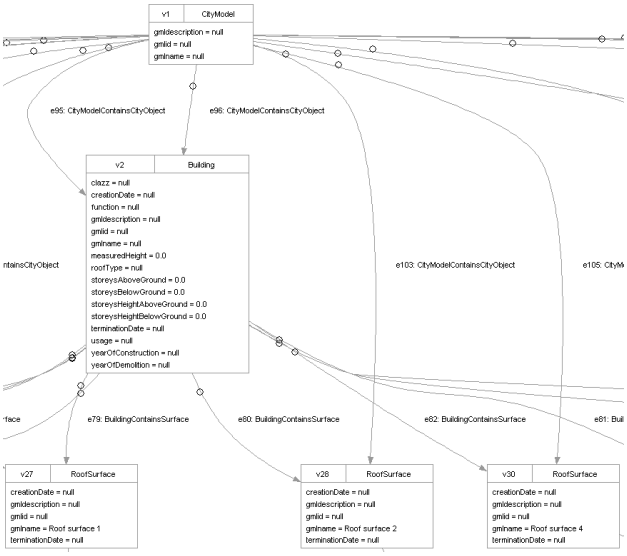

Figure 7: Extraction of the example graph, exported to .dot format and rendered via dotty, a Graphviz tool.

The component `DotModelWriter` writes an integrated model to a .dot-file, the standard file-format of the *Graphviz*[12] graph visualization software (figure 7). The result can be processed further using Graphviz.

The `CityGMLModelWriter` writes the integrated model via a special graph traversal algorithm as a CityGML[Appearance,Building,CityObjectGroup,Generics] model into an .xml-file. The user can influence the result by choosing the LOD and the kinds of textures to be written. The result can be processed further by other tools. For example, it may be rendered via any appropriate CityGML Viewer (see section 1.1). If the exported .xml-model is imported again, information might be lost, since the integrated model contains more information than those covered by CityGML.

[12] `http://www.graphviz.org`

4.6 Model analysis

Many more activities might be implemented on the integrated model, since all kinds of queries may be posed on it. Thus, a further processing activity is the analysis of the integrated model. This activity is not mentioned in the feature diagram in figure 2, since it is carried out as part of nearly every other integrated model processing activity. All tests concerning existing model elements and/or their properties or relationships are *model analysis steps*. There can also be *transversal analyses*, regardind coherences of a whole model, a part of a model (e.g. one building) or a special view to a model (e.g. geometry). Using the TGraph structure traversal and analysis of the integrated model is done repeatedly during runtime using the graph API and/or GReQL queries. Transversal analyses are particularly supported by the graph API. For example it offers iterators for all nodes or edges of a special type (and it subtypes) in the whole model.

Listing 1: Building analyser results.

```
Building 1

Id: 2
Name: Example building
Description: Example building model for testing.

Year of construction: not known
Year of demolition: not known

Number of appearances: 0
Number of building installations: 0
Number of building parts: 0

Number of boundary surfaces: 9
Number of wall surfaces: 4
Number of roof surfaces: 4
Number of ground surfaces: 1
Number of openings: 6
Number of doors: 1
Number of windows: 5

Number of 3d faces: 15
Number of 3d points: 34

Lowest 3d point: Point 1: (0.0, 0.0, 0.0)
Highest 3d point: Point 9: (2.0, 1.0, 4.0)
Height: 4
Width: 4
Depth: 5
Volume: 68
```

To demonstrate the usage of querying with GReQL an additional component `BuildingAnalyser` was developed, that writes information about all buildings of the integrated model into a .txt-file. The file contains different kinds of information. At first there is *semantic attribute information* like name, description and year of construction/demolition of the building. Moreover there is *semantic entity information* like the number of wall, ground and roof surfaces, the number of doors and windows, and so on. Furthermore there are *geometric information* like the count of points and faces of the building geometry or the lowest and highest point of a building. And there is *inferred semantic information* computed using semantic background knowledge in combination with geometry information, like the building height, the building volume, and so forth (listing 1).

5 CONCLUSIONS AND FUTURE WORK

This paper showed how geometric, topological, semantic and appearance information can be integrated in one integrated graph model. The class of models was defined by an *integrated model schema*. Graph representation gives rise to all kinds of algorithmic processing, some examples of which were given, including model creation, improvement, transformation, analysis and export. Using a *lightweight Java component model* some example components were implemented and illustrated based on a simple example.

Though the example has toy character, it should suffice to demonstrate the wide range of manipulation possibilities given by an internal integrated graph representation for the enhancement of urban object models. Since TGraph technology is easily applicable to graphs containing millions of elements, the approach scales to a wide range of applications.

The integrated model was developed in the context of a project for object-recognition (Falkowski et al., 2009). It forms the basis for the application of efficient graph-matching algorithms in this context.

The integrated model schema is still under construction. But it is easily modifiable and each of the three parts can be replaced by different variants. Further goals are the enhancement of the schema for the full *CityGML base profile* (CityGML[full]) and the support for other urban object description languages, like *KML/COLLADA* (section 1.1). Here the tasks are the change and enlargement of the integrated model schema and the adaption of all existing processing components. Some of the described activities could be splitted to more processing steps. A lot of them can be composed to interesting combined processing activities. And there could even be interactive processing components.

Further research topics could be the supplement of more complex model parts to an existing integrated model or the integration of two different integrated models. Another interesting field is the inference of semantics from geometric, topological and/or appearance information.

ACKNOWLEDGEMENTS

This work has been carried out in close cooperation with Peter Decker, Dietrich Paulus and Stefan Wirtz from the Work Group Active Vision as well as Lutz Priese and Frank Schmitt and from the Laboratory Image Recognition, both at the University of Koblenz-Landau.

REFERENCES

Cox, S., Daisey, P., Lake, R., Portele, C. and Whiteside, A., 2001. OpenGIS Geography Markup Language (GML) Implementation Specification. Technical Report 3.1.1, Open Geospatial Consortium, Inc.

Czarnecki, K. and Eisenecker, U. W., 2000. Generative Programming: Methods, Tools and Applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Ebert, J., Riediger, V. and Winter, A., 2008. Graph Technology in Reverse Engineering, The TGraph Approach. In: R. Gimnich, U. Kaiser, J. Quante and A. Winter (eds), 10th Workshop Software Reengineering (WSR 2008), GI Lecture Notes in Informatics, Vol. 126, GI, Bonn, pp. 67–81.

Falkowski, K., Ebert, J., Decker, P., Wirtz, S. and Paulus, D., 2009. Semiautomatic generation of full CityGML models from images. In: Geoinformatik 2009, ifgiPrints, Vol. 35, Institut für Geoinformatik Westfälische Wilhelms-Universität Münster, Osnabrück, Germany, pp. 101–110.

Foley, J. D., van Dam, A., Feiner, S. K. and Hughes, J. F., 1990. Computer Graphics. Principles and Practice. Addison Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Groeger, G., Kolbe, T. H., Czerwinski, A. and Nagel, C., 2008. OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Technical Report 1.0.0, Open Geospatial Consortium Inc.

Herring, J., 2001. Topic 1: Feature Geometry (ISO 19107 Spatial Schema). Technical Report 5.0, Open Geospatial Consortium Inc.

Muller, D. E. and Preparata, F. P., 1978. Finding the Intersection of two Convex Polyhedra. Theor. Comput. Sci. 7, pp. 217–236.

Pohl, K., Böckle, G. and van der Linden, F., 2005. Software Product Line Engineering. Springer, Berlin.