

A Framework for the Data-Driven Analysis, Interpretation, and Transformation of Geospatial Information Models

ANDREAS KRÜGER¹ & THOMAS H. KOLBE²

Zusammenfassung: In Geographischen Informationssystemen spielen Analyse-, Interpretations- und Transformationsprozesse eine zentrale Rolle. In den letzten Jahren wurden räumliche Informationsmodelle wie virtuelle 3D-Stadtmodelle und Katastermodelle, z.B. CityGML, GeoSciML, ALKIS, UK OS MasterMap, etc., verstärkt auf Basis des General Feature Model der ISO191xx Standard-Reihe spezifiziert. Innerhalb dieses Rahmens können Transformationsprozesse auf standardisierten Strukturen mit wohldefinierter Semantik ausgeführt werden. Objektorientierte Modellierungskonzepte erlauben die Spezifikation komplexer Strukturen mit Aggregations- und Generalisierungs-/Spezialisierungshierarchien. Analyse-, Interpretations- und Transformationsoperationen auf räumlichen Datensätzen müssen daher mit der Komplexität dieser Strukturen umgehen können.

In diesem Beitrag wird ein Rahmenwerk entwickelt, welches ein datengetriebenes Transformationssystem auf komplexen Objektstrukturen – repräsentiert als attributierte getypte Graphen – zur Verfügung stellt. Die Verarbeitung dieser Graphstrukturen wird mithilfe von Graphersetzungen realisiert. Es wird eine Transformationssprache vorgeschlagen, die auf der Notation von Geoobjekten und deren räumlichen Bestandteilen arbeitet. Dieses Konzept erlaubt die Erkennung komplexer Konfigurationen räumlicher Geoobjekte durch die Anwendung von Subgraph-Isomorphismen. Des Weiteren wird die Abbildung von Transformationsprogrammen für ein spezifisches Graphersetzungssystem, AGG, sowie eine prototypische Umsetzung beschrieben. Das vorgestellte Konzept ist insbesondere nützlich für Erkennungs-, Interpretations- und Generalisierungsprozesse. Letzteres wird anhand eines Beispiels der Generalisierung von Flurstücken veranschaulicht.

Abstract: In geographic information systems, analysis, interpretation, and transformation of spatial datasets play a central role. In recent years geospatial information models like virtual 3D city models or cadastre models like CityGML, GeoSciML, German cadastre ALKIS, UK OS MasterMap, etc., are increasingly specified on the base of the General Feature Model of the ISO 191xx standard series. Within that framework, transformations on standardized structures with well-defined semantics for geoobjects can be executed. Object-oriented modeling concepts allow for the specification of complex structures with aggregation and generalization / specialization hierarchies. Thus, the analysis, interpretation, and transformation of spatial datasets have to cope with this complexity.

Within this paper, we propose a framework which realizes a data-driven transformation system on complex object structures represented by attributed typed graphs, which can be processed using graph transformation. A transformation language is proposed referring to the notion of geoobjects and their spatial entities. It allows recognizing complex configurations of spatial objects by using subgraph isomorphism. Furthermore the mapping of transformation programs to graph transformation rules for a specific graph transformation system, AGG, and a prototypical implementation is explained. The proposed concept is especially useful for recognition, interpretation, and generalization tasks. The latter is demonstrated for an example on land parcel generalization.

- 1) Andreas Krüger, Technische Universität Berlin, Institute for Geodesy and Geoinformation Science, Strasse des 17. Juni 135, 10623 Berlin, Germany, E-Mail: krueger@igg.tu-berlin.de
- 2) Thomas H. Kolbe, Technische Universität Berlin, Institute for Geodesy and Geoinformation Science, Strasse des 17. Juni 135, 10623 Berlin, Germany, E-Mail: kolbe@igg.tu-berlin.de

1 Introduction

In geographic information systems, the analysis, interpretation, and transformation of spatial datasets play a central role. Since geospatial information models, like virtual 3D city models and cadastre models are defined more and more in a feature-based way according to the General Feature Model of the ISO 191xx standard series, transformation processes on standardized structures with well-defined semantics for geobjects can be defined.

Transformations which affect several related geobjects with topological and geometric relations require a processing of complex object structures including their relationships. Often transformations should only be applied for specific configurations of geobjects. For example, in a generalization task land parcels should only be aggregated, if they have less than a predefined maximum area, have the same landuse type, and their geometries are adjacent. This requires specifying transformation conditions which have to take different geobjects and geometry objects together with their relations into account. Also the recognition and interpretation of geobjects requires matching of (sometimes complex) template structures with a given dataset.

The objects together with their interrelationships can be described by an attributed graph. Nodes and edges are typed according to the corresponding classes of the geobjects or relations. Graph transformation is a practical concept for processing modifications on graphs, but is generally not tailored to express transformations on geospatial datasets. In order to allow for an intuitive specification of transformations, a proper language should take into account the notion of geobjects and their spatial characteristics and relationships. In this paper, we will propose such a transformation approach, where we use graph transformation concepts and suggest a transformation language that is tailored to geobjects.

The rest of the paper is organized as follows: Section 2 discusses geospatial data modeling according to the ISO 191xx standards family. We then show how geodata that is exchanged using the Geography Markup Language (GML) can be mapped to an attributed, typed graph structure and explain general requirements on the transformation of feature-based datasets. Section 3 provides some background on graph transformation and reviews the concepts and capabilities of AGG, the graph transformation tool that is employed subsequently. In section 4 we present a rule-based language for the specification of transformations of geographic features and its translation into the AGG framework, while section 5 briefly describes our prototypical implementation. The example given in section 6 shall demonstrate the principle and potential of our concept. In section 7 we draw the conclusions and give an outlook.

2 Feature-Based Modeling and Transformation

GML and data models conforming to the ISO 191xx standard family can be handled as typed attributed graph structures in the transformation processes. The following subsections give a short introduction to the structure of these data models and the mapping to a graph structure.

2.1 Modeling Standards

More and more geospatial information models are designed according to the *General Feature Model*, *Spatial Schema*, and *Temporal Schema* of the ISO 191xx standards. The *General Feature*

Model describes how real world objects and phenomena can be abstracted and represented by *geographic features* which may have an arbitrary number of spatial and non-spatial properties (ISO 19109:2005). Features are objects belonging to a class, which can have spatial, temporal, thematic, locational, metadata, or quality attributes. Object-oriented modeling concepts allow for the specification of complex structures with aggregation and generalization/specialization hierarchies. Furthermore, arbitrary associations between feature classes are allowed. Thereby, complex geospatial information models or ontologies can be created and formally specified.

For example, *CityGML* is an OGC standard for the representation of virtual 3D city models; *GeoSciML* is a standard for geoscientific information; *ALKIS/ATKIS/AFIS* are the German national standards for cadastre and topography; and *OS MasterMap* is Great Britain's national cadastre model standard. All these standards differ from each other in such respect that they specify their own (complex) feature types and interrelationships with well-defined feature semantics. But they have in common the way how the geobjects are defined, namely as being defined as subclasses from the predefined classes given in ISO 19109, 19107, 19108, and others.

With the *Geography Markup Language* (GML, ISO 19136; see OGC 07-036, LAKE ET AL. 2004) all these standards share the same exchange format. GML is an approved standard issued by the Open Geospatial Consortium (OGC). Due to its conformance to ISO191xx and the General Feature Model we can assume a well-defined structure and semantics of GML-based datasets.

2.2 GML and Graph Structure

The structure constituted by the General Feature Model suggests handling GML datasets as typed attributed graphs (see EHRIG ET AL. 2006 and ERMEL ET AL. 2006). An approach for representing CityGML – a GML application schema – as typed graphs (TGraphs) can be found in (FALKOWSKI & EBERT 2009). However, they do not address the topic of a transformation language. If we interpret GML features as nodes and GML properties and relations as edges, we would get a tree structure between parent and child features connected by properties. Since properties can be realized “by value” or “by reference” (*XLink*), a child can be related to several parent features – it is a graph instead. As an example we can describe a situation, where two features, e.g. a *LandUse* and a *Road* object, refer to the same geometry object. In that case two paths, starting from a shared parent node, will lead to the same geometry node. We may have a cyclic graph, since a feature can be transitively related to itself by reference to features which reference back. Also, the graph is directed. Undirected associations are usually mapped in GML to two directed associations. Complex properties can be realized as additional nodes connected by an object property relation. Atomic properties and attributes can be stored as node or edge attributes. Node types for features and complex properties and edge types for object property relations can be derived from the object-oriented type structure in the XML schema of a GML format. As result, we get a typed attributed graph structure.

In transformation, analysis, and interpretation processes, often more than one GML dataset must be handled. These datasets can be modeled according to different application schemas. We can integrate such models into one graph by insertion of additional topological, geographical, or semantical relations between features. Thus, complex graph structures can occur. In our prototypical implementation (see section 5) we choose the following graph structure:

- Generally, objects (data items which have an object identifier, like features) and complex properties will be stored in nodes with appropriate node types. The following base types are distinguished in GML: *Feature*, *Geometry*, *CoordinateReferenceSystem*, *Envelope*, *TimeObject*, and *Topology*. The types can be determined from XML schema type inheritance. Objects which cannot be assigned to the given types get the node type *AnyObject*.
- Edge types will be given by the kind of relationship, e.g. *ObjectProperty*, *TopologicalRelation*, etc.
- Atomic properties and scalar-valued attributes will be stored as node and edge attributes – the attribution of the graph.

We store a GML geometry structure and its child objects in one single node with type *Geometry*. Within this node we hold geometry as Java objects as defined in the *GeoTools* project API (see *GeoTools-Home*). The *GeoTools* geometry is based on the *GeoAPI* – an official OGC project capturing several OGC standards as Java interfaces (see *GeoAPI*). Thus, we have Java geometry objects, on which geometrical and topological operations built into *GeoTools* can be processed.

2.3 Transformation of Feature-based Datasets

While processing feature-based datasets, we can identify a set of integration, fusion, and transformation cases. A system which provides transformations on feature-based datasets must be able, to handle these cases:

- Analysis of datasets needs enrichment by new properties, attributes, and objects which express analysis results.
- Integration needs join-operations of different feature-based datasets, which provides integration of non-existing objects into another dataset, adjustment calculation on spatially inconsistent objects, and merging of identical objects with adjustment of attributes and properties. A transformation system has to consider topological and geometrical relations.
- Transformation processes have to change objects or complex object structures to other objects or object structures.
- In the conversion of datasets objects can be merged, subdivided or transformed to complex object structures, e.g. if we convert a dataset to another with higher a level of detail.
- Several algorithms need an iterative application of transformations until a given condition is fulfilled, e.g. some generalization approaches.

These examples reveal that processing of feature-based datasets needs to handle simple as well as very complex object structures. By using attributed graph transformation, we can realize a data-driven transformation system on complex object structures presented by graphs.

3 Attributed Graph Transformation

Actually there exist a range of development environments for graph transformation systems, for example, *GReAT* (BALASUBRAMANIAN ET AL. 2006), *PROGRES* (RANGER & WEINELL, 2008 and SCHÜRR ET AL. 1999), and *AGG* (see THE <AGG> HOMEPAGE). These environments differ in notation, underlying paradigms, the implementation language and graph representation. A decision for the used environment depends on the requirements. The *AGG* engine is the best fitting system to our requirements, because we need in our project a platform-independent system for exe-

cution within a Grid Computing environment (see section 5). With AGG we can realize the graph structure explained in section 2.2. AGG is a still supported Open Source development, and it provides a rich set of concepts for typed, attributed graph transformation.

3.1 Attributed Graph Transformation in General

A Graph Transformation is the rule-based modification of graphs. A transformation has the following effects (EHRIG ET AL. 2006):

- The transformation is complete, i.e. all specified changes will be carried out.
- The transformation is minimal, i.e. only the specified changes will be realized.
- The transformation is local, i.e. the changes are affected only to the subgraph of a match.

A rule $p: L \rightarrow R$ consists of a left-hand side (LHS) and a right-hand side (RHS) graph. The LHS graph is a pattern which can match the given host graph. If there is a match with fulfilled application conditions a rule is applicable. Then, in the host graph the LHS will be replaced by the RHS using subgraph isomorphism (MELAMED 1998).

Within graph transformation approaches, nodes and edges can have labels. These labels can or cannot be modifiable. Non-modifiable labels are node and edge types. Modifiable labels used for storing values in nodes and edges are called attributes. In attributed graphs nodes and edges are labeled by a set of attributes. The rule can modify the attributes of nodes and edges of the affected subgraph (EHRIG ET AL. 2006 and MELAMED 1998).

3.2 Attributed Graph Grammar System (AGG)

“AGG is a development environment for attributed graph transformation systems supporting an algebraic approach to graph transformation” (THE <AGG> HOMEPAGE). AGG was developed at the Technische Universität Berlin, Department of Software Engineering and Theoretical Computer Science. The next sections will give an overview about AGG.

3.2.1 The AGG Graph Structure

An AGG graph is a typed attributed graph. The main graph objects in such a graph are nodes and directed edges between two nodes. AGG allows multiple edges between the same pair of nodes. Nodes and edges have an identity of their own. All graph objects are associated with exactly one label, also called type, of a given label (type) set. Attributes of graph objects will be declared by a name and a type. Any value of this type can be assigned to that attribute. Types in AGG can be any valid Java type, i.e. a simple type or a Java class (EHRIG ET AL. 2006 & ERMEL ET AL. 2006).

3.2.2 AGG Graph Transformation Concepts

AGG offers a range of features for the development and application of graph transformations. AGG is platform independent written in Java. It provides a graph transformation engine which can be embedded in Java applications. Transformation rules can be defined and validated, including creation of typed attributed graphs, type graphs with node type inheritance, attribution of graphs by Java objects, types and classes, conditions like Negative / Positive Application Conditions (NAC / PAC), and boolean Java expressions as attribute conditions, attribution of rules by Java expressions, definition of constraints as global graph consistency conditions, and scheduling of rule application by defining layers of rules (AGG-TEAM).

AGG provides the single-pushout approach. There, a rule will be defined by construction of an attributed LHS graph and an attributed RHS graph. Between both sides partial graph morphism: $LHS \rightarrow RHS$ exists. The LHS may contain attributes and constant values. A RHS can additionally contain Java expressions for calculating attribute values. During the rule application, each match of the LHS in a given host graph will be replaced by the RHS. Nodes and edges in LHS with no mapping in RHS will be removed from the host graph. New nodes and edges in RHS will be created. Dangling edges will be removed after a transformation. Attribute expressions on the RHS will be evaluated and calculated. If there is no match, a rule is not applicable. If the host graph contains multiple matches, one will be chosen indeterministically (EHRIG ET AL. 2006 & ERMEL ET AL. 2006). A rule application terminates if no match can be found anymore.

AGG supports the definition of rule layers, which specify an ascending order on a set of rules. Within a layer rules will be applied in an arbitrary order as long as a match can be found. The layer terminates if no containing rule is applicable anymore. A loop over layers can be enabled. In that case the layer sequence will be repeatedly applied, until no layer is applicable anymore. A rule designer has to pay attention to the termination of layers and loop over layers.

A rule is only applicable if the LHS can produce a match. For additional limitation of rule applicability, AGG provides the following instruments to specify conditions (EHRIG ET AL. 2006):

- A set of Negative Application Conditions (NAC) and Positive Application Conditions (PAC) can be added to the rule. If a NAC is satisfied, the rule is not applicable. For given PACs a rule is only applicable if all PACs are fulfilled.
- Attribute conditions are Java expressions with boolean result. By evaluating them, the condition is decided to be satisfied or not.

A developed graph transformation system with AGG can be analyzed by critical pair analysis, consistency checking and termination criteria. The detailed concepts for analyzing a transformation system in AGG can be found in (EHRIG ET AL. 2006) and (AGG-TEAM).

4 Transformation Language

Integration, analysis and transformation problems can be manifold. Thus, it is impractical to hardcode transformation rules. We need a flexible system instead for implementing rules on the basis of a transformation language.

Since we develop a transformation system for GML based, ISO 191xx family conforming models we should assume, that a user of this system is familiar with GIS modelling, but not with graph transformation. So, the transformation language must be a compromise between the power of graph transformation concepts and GIS requirements on transformations in integration, analysis, and transformation cases. The language has to provide the underlying graph transformation concepts but to abstract from them by using structures and addressing of the ISO 191xx family. We need to work with GML object/property structures; not with node/edge structures of graphs. Nevertheless the language must reflect the concepts of graph transformation within AGG. The matching between the transformation language and AGG should be done by an interpreter.

The language concepts of AGG are completely declarative. However, we believe that the formulation of sequential transformation steps provides a more intuitive way to specify transformation actions. Thus, we decided to develop a combined declarative and imperative language concept.

Rules in the transformation language follow the *Event-Condition-Action* (ECA) paradigm – a concept from event-driven architectures and active database systems (GARCIA-MOLINA ET AL. 2002). In our case, *events* are occurrences of a rule’s object structure in a given dataset, *conditions* are preconditions of the rule and constraints, and *actions* are the transformation steps expressed as rule actions. We formulate object structures and conditions in a declarative way; transformation steps called actions will be expressed imperatively. The declarative part specifies object configurations on which the corresponding rules should be applied. The transformation language must be able to handle single objects and complex object structures with object property relations, topological and geometrical relations. Therefore, the transformation language works with object structure declarations instead of single geobjects. A successful identification of the declared object structure (subgraph) within the graph in its current state and the conjoint satisfaction of the condition imply the application of all transformation steps of the rule.

In the following subsections we will describe the structure of a transformation language for the processing, analysis, and integration of ISO191xx family conforming datasets. The given concepts can be mapped onto a concrete language syntax, e.g. in XML format. Since the graph structure is equivalent to the general feature model structure, we can assume that rules expressed on GML objects or object structures are applicable as graph transformation rules on a graph derived from GML datasets and their application schemas.

4.1 Language Structure

The transformation language describes a transformation of GML conforming data models whose instances are represented by a graph. Since we address concrete GML objects or properties with their attributes we have to reference these elements by fully qualified GML names. Thus, firstly we need a declaration of XML/GML namespaces. A transformation program consists of:

- The first part comprises a set of constraints with respect to the GML conforming data model(s). These are general conditions independent of concrete transformation rules which must be fulfilled before and after a rule application.
- The second part will declare at least one transformation rule with its source structure, conditions and processing actions concerning the source structure.
- The last part defines processing instructions for rule application with respect to layers of rules and sequential rule application.

This structure follows the schema of AGG graph transformation programs (c.f. EHRIG ET AL. 2006 and ERMEL ET AL. 2006):

```
transformation [<constraints>] <rules> <rule application>
```

The three parts will be described in the following subsections.

4.1.1 Constraints

The transformation language handles complex object/property structures. Thus, also general constraints for a data model must be able to address such structures. Constraints should provide the same logical operators like rule conditions, with ability to define FORALL and EXISTS conditions on object structures, introduced below. For constraints we propose the following notation:

```
constraint <condition>
```

The following example expresses the general requirement that all Building features in the dataset must have a measured height of greater than 0 meters:

```
constraint forall (∃ Feature citygml:Building b1,  
                  ∃ Variable v1 value $b1/measuredHeight) condition v1>0
```

4.1.2 Rule Definitions

A rule definition has the following structure consisting of four parts:

```
rule <name> <object structure> [<conditions>] <actions>
```

First, a rule needs a name for referencing. Then, two declarative parts of the rule will be given, an object structure, which occurrences should be transformed by the rule and a declarative part defines conditions, if needed. As last part several imperatively expressed actions will be given, which will be applied sequentially on the object structure and create the destination structure.

➤ *Object Structure Declaration*

In order to define an object structure in rules or *EXISTS* and *FORALL* conditions we need a declaration part for objects and relations, as follows:

```
∃ <object type> <namespace>:<classname> <name>  
e.g. ∃ Feature citygml:Building b1,  
     ∃ Geometry gml:Curve c1
```

The declaration requires the object type to identify the object kind, the fully qualified class name, and an object name that acts as variable name for accessing the object. The permitted object types were derived from the GML top-level classes *Feature*, *Geometry*, *CoordinateReferenceSystem*, *Envelope*, *TimeObject*, *Topology*, and *AnyObject* for each other type of an object. Relations are declared analogously:

```
∃ <relation type> [<namespace>:]<classname> <name> from <ref1> to <ref2>  
e.g. ∃ ObjectProperty lod0Network op1 from b1 to c1,  
     ∃ TopologicRelation adjacent tr1 from g1 to g2
```

As in the object declaration we need a relation type, the relation classname and a name that acts as variable. A start and end object of the relation must be given. The type *ObjectProperty* defines a GML property relation. Other relations like topologic and geometric relations should be taken from a predefined set of relation types. Additional to objects and relations, we need variable and Java class instance declarations for later use in statements and method calls of conditions and actions. Similar to other declarations, variable and class instances can be declared as follows:

```
∃ Variable <name> value <XPath expression>  
∃ Instance <name> class <Java Class>  
e.g. ∃ Variable v1 value $b1/measuredHeight  
     ∃ Instance i1 class java.lang.String
```

Variables and instances need a name for access. The value of the variable can be objects or attribute values, specified by an *XPath* expression which refers to the data model of the dataset, i.e. its GML application schema. So the developer of a transformation program is not required to know the graph representation but only the structure of the GML application schema. The variable type will be automatically determined from the type of the referred object/value. The instantiated Java class will be specified by a fully qualified Java conforming notation. All given names

can be used as variables within all parts of the configuration language, also in Java and *XPath* expressions. Similarly, declared Java variables are visible in non Java parts. In *XPath* notation we have to precede variable names by a '\$' due to *XPath* conventions.

➤ *Conditions*

When the declaration of objects and relations is done we have the object structure which will be translated to the left hand side graph of the rule. Each occurrence of this structure in the host graph would produce a match. Preconditions define additional restrictions for a rule application:

```
precondition <condition>
```

In order to express conditions we need a set of logical operators, like *AND*, *OR*, *FORALL*, *EXISTS*, *NOT*, *<*, *>*, *=*, etc. Some of these operators and conditions can be expressed by boolean Java method calls as like in AGG. *FORALL* and *EXISTS* are used for specifying conditions on graph object structures. So they need an object structure declaration. Additionally, the operators *AND*, *OR*, and *NOT* should be provided in the language for formulation of complex conditions:

```
not <condition>
<condition> and | or <condition>
forall <object structure> <condition>
∃ <object structure> [<condition>]
```

Composed conditions can be enclosed by parentheses in order to modify the interpretation sequence. With logical operators we can formulate simple and complex conditional expressions, e.g. a variable must have a value between 1 and 25 or two different geometries must be adjacent:

```
precondition not (v1<1 and v1>25)
precondition forall (
    ∃ Geometry gml:Curve c1,
    ∃ Geometry gml:Curve c2,
    ∃ TopologicRelation adjacent from c1 to c2) condition not (c1=c2)
```

➤ *Actions*

Actions are imperative commands, i.e. sequentially ordered instructions to incur changes on the declared object structures. With that, each modification step can be expressed separately in a specific order. Three kinds of instructions are provided to formulate actions: a *CREATE* operator for object and property creation, a *DELETE* operator for objects and properties, and a *SET* operator for setting attribute values. The operators should have a structure as follows:

```
create <object type> <namespace>:<classname> <name>
create <relation type> [<namespace>:]<classname> <name> from <ref1> to <ref1>
delete <ref>
set <XPath expression> value <Java expression>
e.g. create Feature citygml:Building b1
      set $b1/measuredHeight value 10
      delete b1
```

The structure of the *CREATE* operator corresponds to the object structure declaration. The *SET* operator references attributes at a given *XPath* location in the GML dataset. The variable type must match the attribute type.

4.1.3 Rule Application

A rule application declaration is a processing instruction that configures the scheduling of the declared rules. If no rule application is defined, we assume an indeterministic application of the

given rules in a single layer. This formulation provides the AGG concept of using layers in the conceptual transformation language:

```
ruleapplication loop <true|false> <layers>
```

There, a set of layers containing several rules will be defined. The loop setting defines whether a loop over layers is enabled. In that case the layer sequence starts again after application of all layers, until no layer is applicable anymore. Similarly to AGG, a layer terminates if no rule can be applied anymore. Layer and rule references have to be given as follows:

```
layer <number> [terminationcondition <condition>] <rule references>
rule <ref>
```

A layer should have an arbitrary layer number, which is used to define the layer order. A set of termination conditions can be given for additional restriction of layer application. Termination conditions will be defined by using the same notation like rule conditions. Finally, a layer references a set of defined rules. An example rule application declaration can look as follows:

```
ruleapplication loop true
  layer 0
    terminationconditions
      forall (∃ Geometry citygml:Building b1,
              ∃ Variable v1 binding $b1/measuredHeight) condition v1>0
      rule r1 rule r2 rule r3
```

In the next section we will explain how the specified transformation language concepts are translated to proper graph transformation rules for AGG.

4.2 Translation of Language Concepts to an AGG Graph Transformation

Since the transformation language contains declarative and imperative parts, but rules in AGG are defined completely declarative, we must translate a program given in the transformation language to AGG rules by interpreting the language parts.

The translation of constraints depends on its domain. If constraints restrict the object structure we map them to AGG graph constraints. Constraints which restrict attributes will be transferred to conditions of the LHS and RHS. Java expressions can be adopted, since AGG can handle such expressions. Other logical operators must be interpreted and mapped to adequate operators.

Each rule given in the transformation language is translated into a rule in AGG. The object structure declaration will be translated to the LHS graph of the rule. There, the object type decides about the node type of an object. Relation types will decide the edge type of a relation. Variables and Java class instances are mapped to variables of the AGG rule. Preconditions which affect the object structure will be mapped to a modified LHS graph acting as additional NAC or PAC. Preconditions which affect attributes will be translated to conditions of the rules LHS. Actions are imperative instructions which must be sequentially executed on the LHS graph. All instructions will be interpreted and processed, which results to a target – the RHS graph within the AGG rule. Appropriate attribute matchings will be created. All *XPath* expressions will be resolved.

The rule application declaration and layers with rules will be mapped from the transformation language to layers of the AGG transformation. Subsequent layers will get an ascending ordered layer number which defines the processing order. Rule assignments to a layer will be retained. An enabled loop over layer configuration will be adopted within AGG. Termination conditions of a layer will be translated to conditions of each rule in this layer.

5 Prototypical Implementation

We have implemented a prototype by using AGG, the graph structure introduced in section 2.2, and the transformation language. The following program flow is realized, illustrated in Fig. 1:

- For the generic handling of GML models we use a reader that can read type information from GML application schemas given as XML schema files. This information contains the structure of declared data types. Based on this information the reader can read GML files of different types and convert them into the internal representations of GML objects and properties.
- For further realization we use AGG. Type information will be mapped to node and edge types. These are the basis for creation of nodes and edges. GML objects and properties will be read in by checking the conformance to its type information and using *GeoTools* (see *GeoTools-Home*) for creating geometry nodes. The nodes and edges form the whole source graph in AGG.
- By translation of a given transformation file conforming to the formalized transformation language we derive a set of graph transformation rules and a schedule for rule application. By using the AGG engine the transformation rules will be applied on the source graph. After termination we get the target graph.
- Finally, the target graph will be resolved to target GML objects and properties and the appropriate type information. The writer can now create the resulting GML file.

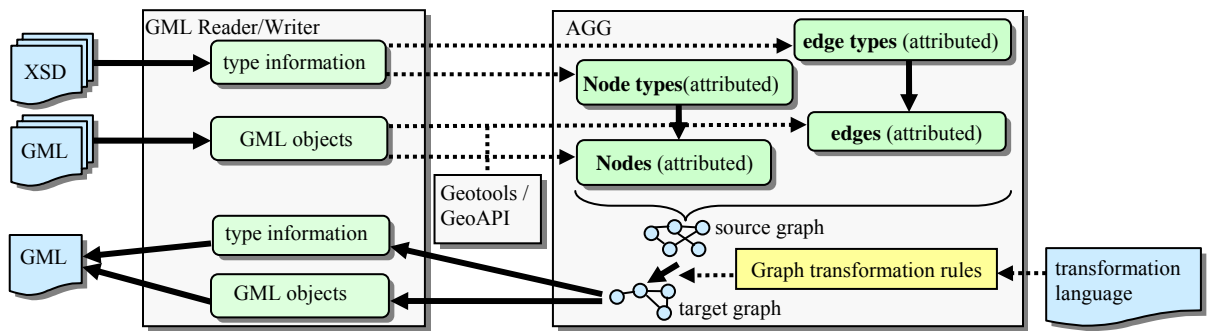


Fig. 1. Prototypical Implementation - overview

With this program structure we can generate files with more than one target GML application schema. Thus, the program can produce several output models. *GeoTools* includes the *Java Topology Suite (JTS)* and is based on the *GeoAPI*. By using *GeoTools* we can use geometrical and topological operators of *JTS* and *GeoTools*. Additionally we have an OGC conforming geometry object structure. For optimized processing of geospatial information models we need a set of Java methods which provide geospatial functionality within Java helper classes.

Everything is implemented in Java and can thus be also executed on Grid Computing nodes. In fact, we are using the implementation within a research project called GDI-Grid (Spatial Data Infrastructure Grid). This project focuses on efficient integration and processing of geodata based on Geo Information Systems (GIS) and Spatial Data Infrastructures (SDI) by grid-enabling OGC compliant web services for grid environments. For that, scenarios of noise dispersion simulation and disaster management will be realized. The here proposed approach of realisation of transformations by using graph transformation will be used within a 3D generation service as

basis of noise dispersion calculation. This service will generate 3D CityGML models from GML based 2D cadastre models and 2.5D Digital Terrain Models. More about this research field and the GDI-Grid project can be found in (KRÜGER & KOLBE 2008 and WERDER & KRÜGER 2009).

6 Application Example

In order to illustrate the graph representation and formulation of transformations on such graphs, we will demonstrate the development of a transformation program in a short example. The example realises a method of map generalization: the aggregation of areal objects from one scale to a smaller scale formulated as an optimization problem using combinatorial optimization techniques, developed in (HAUNERT 2008). This method should generate results of maximal data quality and is weighted, e.g., by a minimal area size. Criteria for data quality are:

- Aggregated objects should be members of semantically similar classes (like greenland, settlement area, etc.), formalized by a semantic distance between classes.
- A composition of objects should have a geometrically simple and compact shape.

The algorithm given in (HAUNERT 2008) is based on an adjacency graph of a planar subdivision, where a node represents a region and is weighted; edges between two nodes represent a common boundary between two regions. After application of the area aggregation to a smaller scale, a given weight threshold must be exceeded. The algorithm has to find a partition of the given graph nodes and a set of classes, such that:

1. The adjacency graph of all areas within the partition is still connected.
2. A node of the areas within the partition has an unchanged class – called the *center*.
3. Each area within the partition has a total weight of at least the weight threshold.
4. A cost function dependent on total cost for class change and total cost for non-compact shapes of all subareas is minimized. The cost for non-compact shapes will be given by an appropriate cost function, which is a product of weight and semantic distance.

Fig. 2 shows the file structure of a planar subdivision of five CityGML areas of class *Cropland*.

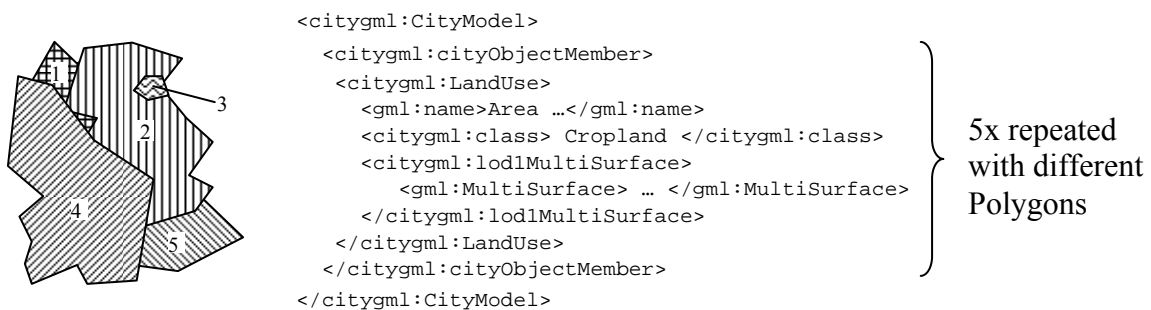


Fig. 2. Example of a simple CityGML instance document with 5 areal LandUse objects

This CityGML file leads to a graph (in this example a tree), as shown in Fig. 3. Each *LandUse* feature in the feature collection *CityModel* will be mapped to a feature node. The geometry of each feature will be stored encapsulated in an own geometry node. The object property relations will be represented by directed edges between nodes. Topologic relations – in this case adjacency relations – have to be added for each pair of adjacent geometry objects before the application of the algorithm. So, an explicit adjacency subgraph is added to our graph, shown in Fig. 3.

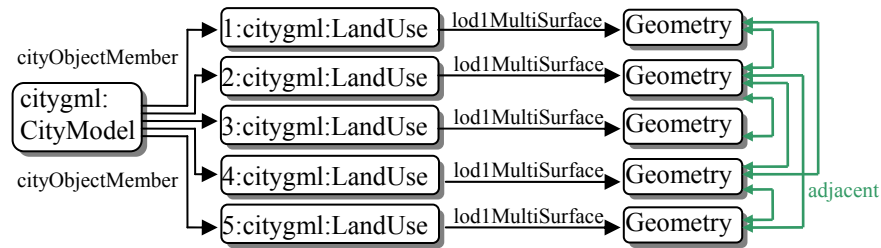


Fig. 3. Simplified graph with inserted adjacency relations

In this graph we can realize the aggregation of areal objects as described above using graph transformation. For that purpose, we map the four generalisation rules given above to a transformation program according to our language. As stated before, a rule consists of two declarative parts - object structure and condition declarations - and an imperative definition of actions. The source object structure of aggregation of two *LandUse* features can be declared as:

```
rule area_aggregation
  ∃ Feature citygml:LandUse f1,
  ∃ Feature citygml:LandUse f2,
  ∃ Geometry gml:MultiSurface g1,
  ∃ Geometry gml:MultiSurface g2,
  ∃ ObjectProperty citygml:lod1MultiSurface p1 from f1 to g1,
  ∃ ObjectProperty citygml:lod1MultiSurface p2 from f2 to g2,
  ∃ TopologicRelation adjacent r1 from g1 to g2,
  ∃ TopologicRelation adjacent r2 from g2 to g1,
  ∃ Variable n1 value $f1/name,
  ∃ Variable n2 value $f2/name,
  ∃ Variable c1 value $f1/class,
  ∃ Variable c2 value $f2/class,
  ∃ Variable ms1 value $g1/geometry,
  ∃ Variable ms2 value $g2/geometry,
  '-- the following line refers to a user defined class: --
  ∃ Instance ah class de.gdigrid.transform.AreaHandler
```

For later use, we define a set of variables and an instance of a handler class, which provides several Java methods for calculations on the areal geometries.

The declaration of conditions must reflect the four construction conditions of the algorithm given above. The first condition of connectivity of composed areas is already implicitly fulfilled by the object structure declaration. As class we take the *class* attribute of a *LandUse* area. The second condition is also implicitly fulfilled since the class remains unchanged. In this example the size of an area object acts as weight. We need a termination condition for the continuous rule application while matches exist. Here it is given by exceeding the total weight threshold – a minimal area size – of the composed area. After the termination of the continuous rule application, we can guarantee the third condition if the weight threshold isn't too high. Finally, we need a cost function dependent on the semantic difference between two areas, a total weight and a compactness function of both geometries, which must be minimal to fulfil the fourth condition. Details on the cost function are given in (HAUNERT 2008). Since we can use Java expressions we define the conditions by using Java methods provided by a declared *AreaHandler* instance:

```
precondition ah.getArea(g1) < 100 and
precondition
  ah.costFunction(ah.semanticDifference(c1,c2), ah.getWeight(g1,g2),
  ah.getCompactness(g1,g2))<=30
Please note: We define 30 as acceptable minimum value of the cost function to
simplify the example. The weight threshold in this example is set to 100 km2.
```

Now, we can formulate the imperative instructions – a union of the areal geometries and a concatenation of the names of the two affected *LandUse* features. The instructions stepwise lead to the destination graph structure:

```
set $f1/name value n1+n2      '-- string concatenation --
set $g1/geometry value ah.union(ms1,ms2)
delete f2, delete g2, delete p2, delete r1, delete r2
```

The rule can be translated to AGG, as shown in Fig. 4 according to the concepts in section 4.2.

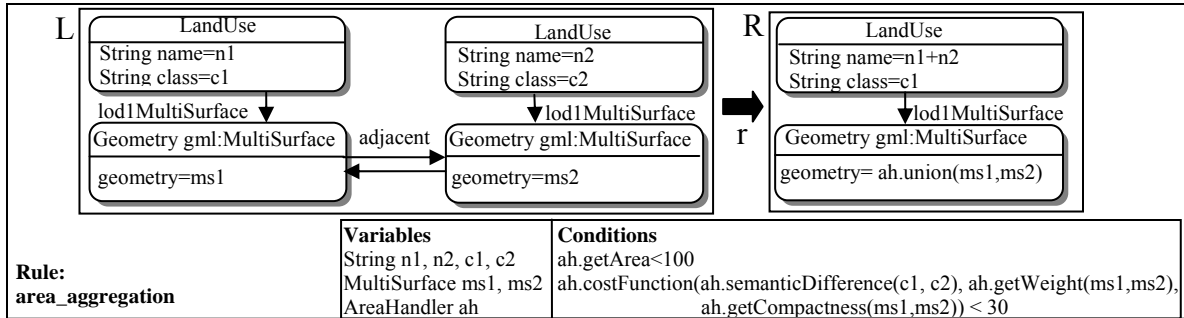


Fig. 4. AGG rule of the example algorithm, notation according to (MELAMED 1998 and EHRIG ET AL. 2006)

For the one defined rule *area_aggregation* we can have a simple rule application with a layer containing the one rule, mapped to an AGG layer including the rule in Fig. 4.

```
ruleapplication loop=false
layer layer1
rule area_aggregation
```

Now we can apply the example algorithm as rule within AGG. The rule will be stepwise applied to the given host graph until no match can be found, either by a non-fulfilled condition, e.g. exceeding weight threshold of all areas, or by none occurrence of the left-hand side graph structure within the host graph. The rule can be applied once, many times or not at all.

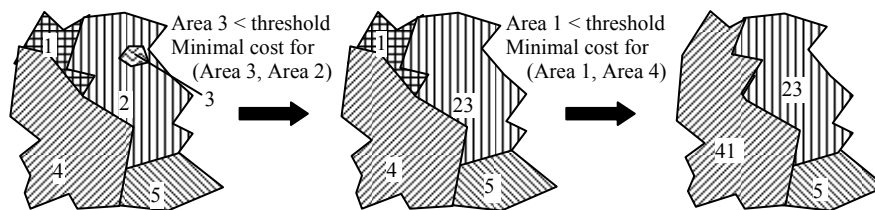


Fig. 5. Two applications of the *area_aggregation* rule

Fig. 5 depicts two applications of the *area_aggregation* rule. Here we can see the stepwise aggregation controlled by a cost function and a weight threshold. This example shows the data driven processing of the transformation by subsequent application of graph transformation rules, since the occurrence of matches – depending on the underlying data – decides if a rule is applicable or not. New matches can result from a rule application; other existing matches can become abolished. The rule terminates only if no match is left.

7 Conclusions & Outlook

In this paper we have proposed a framework for the analysis, interpretation, and transformation of arbitrary spatial datasets modelled according to the ISO 191xx standards and being physically

encoded in GML. We have shown how complex geospatial information models are mapped to attributed, typed graphs. The set of predefined node and relation types directly follows from the ISO 191xx standards and, thus, have specified and standardised semantics. The graph representation maintains the individual property relations between geographic feature instances and their geometric and topologic objects, aggregation relations between features, geometric and topologic relations and generic associations between spatial objects.

This representation facilitates the application of graph transformations. By using the graph transformation tool AGG, transformation rules can be specified in a declarative way. Complex spatial configurations can be identified using AGG's mechanism for subgraph isomorphism matching. This makes the proposed framework especially useful for recognition, interpretation, and generalization tasks. Also ontological transformations can be expressed by the creation of nodes which represent features according to a different application data model (e.g. representing a different ontology) given by a different GML application schema.

Furthermore, we have introduced a transformation language that copes with processing of complex GML object structures and provides concepts for graph transformation in an intuitive way. The notation is partially declarative concerning the expression of object structures and conditions and partially imperative concerning the expression of sequential transformation actions. The declarative part enables the recognition of complex substructures; the imperative actions provide a way to describe specific transformations on them. Since all feature types are fully qualified with respect to their namespaces, datasets from different GML application schemas can be combined and transformed in an integrated way. Because geometries comply with ISO19107 and GML, we can employ the *GeoAPI* framework and the *GeoTools* in our implementation to store and analyse geometry objects.

Based on this work, we see several further research directions. We want to realize more complex transformations within the above stated GML graph structure. It is planned to realise the recognition of bridges spanning valleys on the basis of GML based 2.5D Digital Terrain Models and GML based 2D cadastre data (German ATKIS and ALKIS). Therefore, we have to specify appropriate rules and implement accordant Java methods within the presented prototype. Another task for future research is the analysis of limitations of this concept, which can be named as:

- The application of subgraph isomorphism is NP-complete. On large graph structures this concept produces high computational cost. Thus, we have to analyze the cost-value-ratio in comparison to other transformation engines for geospatial information models.
- Since rules will be executed in arbitrary order on the same graph structure a rule must not reverse a transformation of a concurrent rule, otherwise a termination problem occurs.

The incorporation of inexact subgraph isomorphisms for rule matchings is a further research task aiming at the handling of uncertainty and incomplete or erroneous spatial datasets.

Acknowledgments

This research has been carried out within the framework of the GDI-Grid project as part of the German D-Grid initiative. It was funded by the German Federal Ministry of Education and Research (BMBF). We thank the BMBF for supporting our work.

References

- AGG-TEAM. The AGG 1.5.0 Development Environment: The User Manual. URL=<http://user.cs.tu-berlin.de/~gragra/agg/AGG-ShortManual/AGG-ShortManual.html>. Access: 2010-01-13.
- BALASUBRAMANIAN, D., NARAYANAN, A., VAN BUSKIRK, C., KARSAI, G. 2006. The Graph Rewriting and Transformation Language: GreAT. In: Proceedings of the 3rd Int. Workshop on Graph Based Tools, ECEASST, Vol. 10
- EHRIG, H., EHRIG, K., PRANGE, U., TAENTZER, G. 2006. Fundamentals of Algebraic Graph Transformation. Springer, Berlin et al.
- ERMEL, C., RUDOLF, M., TAENTZER, G. 1999. The AGG approach: language and environment. In Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools, G. Rozenberg, Ed. World Scientific, Singapore et al., 551-603.
- FALKOWSKI, K.; EBERT, J. 2009: Graph-based urban object model processing. In: Object Extraction for 3D City Models, Road Databases and Traffic Monitoring - Concepts, Algorithms and Evaluation (CMRT). ISPRS. 38 - 3/W4, 115-120.
- GARCIA-MOLINA, H., ULLMAN J.D., WIDOM, J. 2002. Database Systems: The Complete Book. Prentice Hall, Upper Saddle River, New Jersey.
- GeoAPI. URL= <http://geoapi.sourceforge.net/charter.html>. Access: 2010-01-19.
- GeoTools-Home. URL= <http://geotools.org>. Access: 2010-01-21.
- HAUNERT, J-H. 2008. Aggregation in Map Generalization by Combinatorial Optimization. Doctoral Dissertation. Institut für Kartographie und Geoinformation, Universität Hannover
- ISO 19109:2005:Geographic Information – Rules for application schema, 2005. International Organization for Standardization, Genf.
- KRÜGER, A., KOLBE, T.H. 2008. Mapping Spatial Data Infrastructures to a Grid Environment for Optimised Processing of Large Amounts of Spatial Data. In ISPRS 2008: Proceedings of the XXI congress: Proceedings of Commission IV. Beijing, 1559-1564.
- LAKE, R., BURGGRAF, D.S., TRNINIC, M., RAE, L. 2004. Geography Markup Language (GML): Foundation for the Geo-Web. John Wiley & Sons, Chichester, England.
- MELAMED, B. 1998. Design and Implementation of an Attribute Manager for Conditional and Distributed Graph Transformation. Master Thesis. Technische Universität Berlin.
- OGC 07-036: OpenGIS® Geography Markup Language (GML) Encoding Standard version 3.2.1. 2007. C. Portele Ed., Open Geospatial Consortium Inc.
- RANGER, U., WEINELL, E. 2008. The Graph Rewriting Language and Environment PROGRES. In: Applications of Graph Transformations with Industrial Relevance, vol. 5088 of LNCS, Springer.
- SCHÜRR, A., WINTER, A. J., ZÜNDORF, A. 1999. The PROGRES Approach: Language and Environment. In Handbook of graph grammars and computing by graph transformation:vol.2: applications, languages, and tools, G. Rozenberg, Ed. World Scientific, Singapore et al., 485-550.
- The <AGG> Homepage. URL=<http://tfs.cs.tu-berlin.de/agg>. Access: 2010-01-14.
- WERDER, S., KRÜGER, A. 2009. Parallelizing Geospatial Tasks in Grid Computing In: GIS.SCIENCE, 3/2009, abcverlag, Heidelberg, 71-76.