Efficient Conceptual Schema Translation for Geographic Vector Data Sets

Thorsten Reitz[1], Ulrich Schäffler[2], Eva Klien[1], Daniel Fitzner[1]
[1] Fraunhofer Institute for Computer Graphics Research, D-64283 Darmstadt
[2] Technische Universität München, D-80333 München
thorsten.reitz@igd.fraunhofer.de, schaeffler@tum.de, eva.klien@igd.fraunhofer.de,
daniel.fitzner@igd.fraunhofer.de

INTRODUCTION AND MOTIVATION

By transforming geographical data sets with heterogeneous conceptual schemas to a harmonized schema, such as the ones provided by the INSPIRE drafting teams, these data sets effectively become one big virtual data set (Zhao et al. 2008). Consequently, Conceptual Schema Translation (CST), i.e. the transformation of a data set that adheres to the modeling of one conceptual schema to another, plays a central role in achieving data interoperability in the context of spatial data infrastructures.

However, achieving efficient CST in terms of hiding complexity of the process from users and the performance of the actual data set transformation is a topic of on-going research (Cruz et al. 2007; Fan & Xiao 2009; Lutz et al. 2009). We present an approach that allows to use a compact, declarative mapping language and also to achieve high-performing geographical data set transformation. The following two aspects form this paper's contribution:

- Adoption of an executable, declarative and compact language for conceptual schema mapping to complex geospatial data integration/harmonisation scenarios
- Development and implementation of an algorithm for the efficient execution of such declarative mappings in the aforementioned geospatial data integration scenarios

EXPRESSING A SCHEMA TRANSLATION

Expressing the rules for a schema translation can be very complex. As an example, the implementation of a three-page matching table in the form of a set of XSLT operations can easily yield a script with thousands of lines (Letho & Sarjakovski 2004). These scripts are hard to create, and even harder to maintain. Therefore, we investigated methods to declaratively define how the elements of a given source and a target schema relate to each other, leaving as many internal details of the transformation execution as possible to the system. Because of its expressivity and compactness, we chose to define translations using the Ontology Mapping Language (OML) (Scharffe et al. 2008). Despite the "Ontology" in its name, it can easily be used with different conceptual schema languages, such as GML Application Schemas, UML or Database Schemas. To account for specific requirements in transforming geospatial data, we created a profile of that language dubbed the geographic OML (gOML). Within this profile, it is for example possible to use OGC Common Query Language (CQL) expressions to filter the scope within which a mapping is to be executed.

In the gOML, a mapping between two schemas is called an *Alignment*, whose main elements are called *cells*. A cell always aligns a single source *entity* and a single target *entity*. Such entities can be single classes, attributes, relations or instances, as well as complex expressions built from these, e.g. conjunctions or unions of all of the above. For each entity, a transformation can be defined that identifies a black-box function that should be invoked to transfer the source entity's content to the target entity. Furthermore, restrictions can be expressed on the allowable content of each entity (called value conditions) and on the allowable types (type conditions). The listing below gives an

example for a cell that maps features of one type (`Watercourses_VA`) fulfilling a value condition to become features of a different type (`Watercourse`).

```
<align:Cell>
  <omwg:entity1>
    <omwg:Class rdf:about="http://esdi-humboldt.eu/Watercourses_VA">
      <omwg:transf rdf:resource="RetypeFeatureFunction" />
      <omwg:attributeValueCondition>
         <omwg:Restriction>
            <goml:cqlStr>LEVEL == 1</goml:cqlStr>
         </omwg:Restriction>
      </omwg:attributeValueCondition>
    </omwg:Class>
  </omwg:entity1>
  <omwg:entity2>
    <omwg:Class rdf:about="hy-p:Watercourse"/>
  </omwg:entity2>
</align:Cell>
```

OML alignments can also be used to perform tasks such as ontology mediation, merging or class level reasoning. When using them for instance translations, some specific rules have to be taken into account for the definition of the mappings. Consider the example of a single feature type `Waterbody` in the source schema being mapped to two feature types `Lake` and `River` in the target schema. Using the constructs of OML for expressing such relationships, one would map `Waterbody` via a *subsumes* relationship to the union of `Lake` and `River`, expressing that the union of the extensions of `Lakes` and `Rivers` is a subset of the extension of `Waterbody`. For the instance transformation, this is not sufficient since a rule specifying which instances should become `Lakes` and which `Rivers` needs to be specified. Hence, we define such mappings in individual cells with entity1 and entity2 being simple (non-composed) feature types, instead of using unions.

A special case centers on instances of multiple feature types being merged. Here, the OML constructs for expressing complex classes or attributes are relevant. Instance merging is represented in a single OML cell defining a mapping between the composed source feature types (in OML: the conjunction of the source classes) and the target feature type. An integral part of such a cell is a merge condition (such as a foreign key dependency or a specific spatial relationship) on the instances. The merge condition is defined as a transformation parameter using a small Domain-Specific Language (DSL).

EXECUTION OF A GOML MAPPING

The key to acceptable performance in the execution of a mapping lies in the analysis of the mapping and in the ordering and grouping of features based on this mapping analysis. Therefore, as a preparatory step, the cells declared in the OML document to be used in execution are analyzed and classified. Furthermore, two special types of cells, cells with filters and cells that define augmentations are registered.

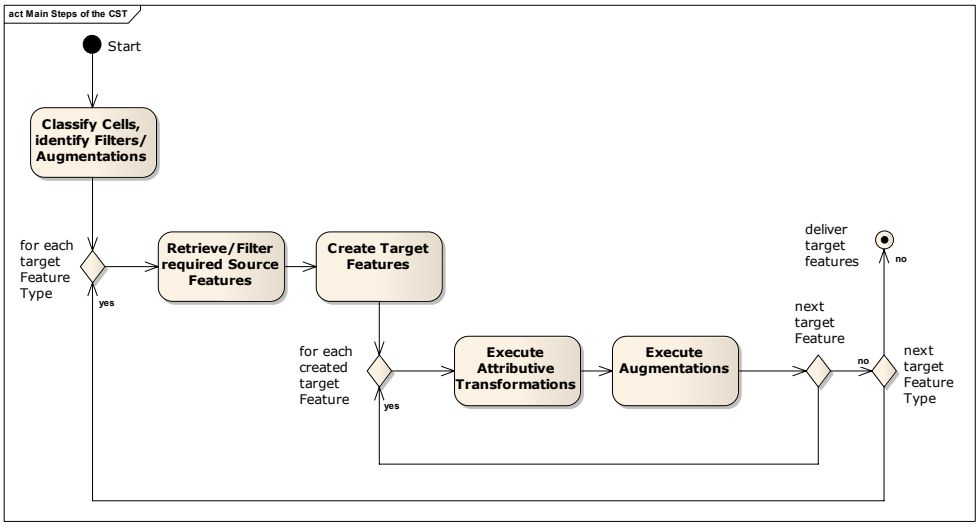

Figure 1: UML Activity diagram providing an overview of the CST execution process

Following this step, actual execution can take place. Execution is split into multiple passes, where in a first pass all required target features are created. In a second pass, attributive transformation functions are carried out, and in the final and third pass augmentation functions are executed. The next sections first describe how different cell types are classified according to execution requirements and then show in detail how the execution is carried out.

Feature Type Cardinality

As described above, a single OML cell specifies a mapping between two entities. There are two main types of cells: those where entity1 and entity2 are Feature Types (or composed Feature Types), in the following called Feature Type cells. Further, there are cells where both entities are attributes, in the following called attribute cells. The case of a single source Feature Type to be mapped to a single target Feature Type is specified in a single OML cell with entity1 being the single source-, entity2 the single target Feature Type (we call such a cell a **1-to-1 Feature Type cell**). Further, for each attribute that shall be mapped, individual attribute cells must be defined (for a detailed description of the OML cells for attribute mappings, please refer to *Table 1*). We call this case with a 1-to-1 Feature Type cell a 1-to-1 mapping.

In addition to such simple 1-to-1 mappings, there can be mappings that map multiple Feature Types in the source schema to a single Feature Type in the target schema and vice versa. As long as no instance joins over multiple source types are used, such cases can be represented by simple 1-to-1 mappings as described above. Specifically, a mapping of *n* source Feature Types to a single target Feature Type (n-to-1) can be represented by n 1-to-1 mappings (and hence n 1-to-1 Feature Type cells plus the attribute cells) as described in the previous section. Further, a mapping of a single Feature Type to n target Feature Types can similarly be represented by n 1-to-1 mappings, using filter expressions to define which subset of the instances of the source FeatureType will be retyped to the different target Feature Types. The following table provides a classification of the cardinality of attribute mappings that can occur.

Attribute Cardinality:	**Explanation:**
1 – 1	A single OML cell defining a mapping between a source and a target attribute.
1 – n	n OML cells defining a 1 – 1 mapping for each

	target attribute or a single cell that defines how to split an attribute value to n attribute values.
n – 1	1 OML cell defining the mapping via a transformation function with n arguments (i.e. attribute names in the mapping, attribute values in the execution) and a single output.
n – m	m OML cells, each defining a n – 1 mapping.

Table 1: Attribute Cardinalities

Instance Cardinality

The previous section described mappings in which a single instance of the source data set should become a single instance in the target data set. However, when the types mapped in the source and target schema are of a different granularity, there is a need to specify *instance splits* (1-n cases) or *merges* (n-1 cases). Instance cardinalities are fully **orthogonal** to the attribute cardinalities as given in *Table 1* in the sense that the instances are merged / split independent of the attribute mappings. When multiple instances are to be merged to a single target instance, they are merged first based on defined merge conditions and subsequently the resulting instance is handled as a usual (non-merged) instance. Vice versa, if one instance shall be split into multiple, the splitting occurs first based on the split condition and subsequently, the generated instances are handled similar to native (non-split) instances. For a description of where the merge / split conditions are specified (i.e. in which cells), please refer to *Table 2*.

Instance merges over multiple source feature types, also known as *joins*, are more complicated. We define joins in a single OML cell with the source entity, *entity1,* being a composed class (i.e. a class conjunction in OML terminology) of the source Feature Types that additionally contains a merge condition. This case is in principle similar to a merge of instances over a single Feature Type, except that the merge condition refers to attributes of multiple feature types instead of only one. The subsequent mapping of attributes is similar to a 1-to-1 mapping between feature types except that the source attributes are the union of all attributes of all source feature types. The merge condition must in general contain the attributes on which the merge should be based on and the aggregation function according to the attribute types.

Vice versa, instance splits to multiple target Feature Types need no explicit representation in OML. Such cases can occur since the mappings do not need to be exclusive. In case of a single feature type in the source schema is mapped to multiple (n) Feature Types in the target schema, this is represented by n OML-Feature Type cells. In case, such cells do not contain filters (or they contain filters that are not exclusive, i.e. there can be source instances satisfying multiple filters), a single instance of the source FT becomes multiple instances of different Feature Types in the target schema.

Instance Cardinality:	**Explanation:**
n - 1	This situation describes an instance merge of multiple instances in the source schema to a single instance in the target schema. This is expressed in the single OML cell specifying the FT mapping e.g. via a construct similar to an alpha-numeric *group-by* clause or a merge of features defined by a spatial relationship or spatial aggregation function (e.g. merge by a bounding-box).
1 - n	This case describes a mapping that splits a single source instance to multiple instances in the target schema. Typically, such instance splits would be

	specified within the cell specifying the FT mapping.

Table 2: Instance Cardinalities

Filters

In principle, any Cell may contain conditions („Filters"). However, the meaning differs depending on whether a cell specifies a mapping between feature types or attributes. A filter within an OML cell specifying a mapping between feature types simply specifies a condition on the source feature types to be mapped and typically filters the source features. It specifies for which subset of source features the mapping shall be executed and typically contains a restriction on attribute values of the source features.

A filter specified on an attribute mapping specifies the source features on which the attribute mapping shall be executed. Logically, such a filter should specify a subset of the instances selected by a filter at feature type level.

For any single transformation, there may be only one filter. This filter can contain multiple conditions, though. Filters are modelled using Restrictions with CQL (the OGC Common Query Language) strings, as in the example provided above.

In OML, it is also possible to construct Filters by using Restrictions without CQL strings. These have to be transformed to CQL strings by the CST to have the possibility to use just one Filter implementation. Only the subset of the input FeatureCollection matching the Filter actually has to be transformed and is passed to the transformation functions.

Augmentation Functions

Augmentation functions are used to define values for attributes of Features in the target schema where no value could be retrieved from a normal transformation, e.g. because respective data was not available in source features. A typical case involves setting attributes to default values, but it can also be tried to derive values from other values that have been predefined. Augmentations can either be functions with no parameters (i.e. constants), functions with constant parameter values specified by users during the mapping (constant in the sense that they do not depend on / change with source instances) or with parameters values derived from already transformed values (target values). Hence, since augmentations are independent of source data and might depend on already generated target values, they are, opposed to Filters, executed at the very end of the transformation – all other transformations that are important for a Feature of `FeatureType` T_A will have to be executed beforehand.

The Main Transformation Process

Most transformations between feature types can be represented by cells with *entity1* and *entity2* being single (non-composed) feature types. The only case that is not expressible with such 1-to-1 mappings is the instance merge over multiple source feature types. In such case, *entity1* is a composed feature types or, in OML terminology, the **conjunction** of multiple source feature types. In the following, we give an algorithm for executing OML mappings (i.e. sets of OML cells) for all of the above cases.

The input to the execution algorithm is a set *C* of OML cells specifying a mapping between two schemas as well as a set I of instances of the source schema. Let m <= |C| be the number of OML cells specifying a mapping on FT level (i.e. FT cells). The following algorithm iterates over all (m) OML FT cells in the mapping and executes them for each source instance of the source FT. Several cases need to be distinguished, e.g. whether the source FT is a composed feature type, whether the cell contains a merge condition etc. These details are given below.

For each OML cell *C* with *entity2* being a Feature Type (target):
Initialize empty set of entity1 instances *entity1Instances*
If *C* contains a merge condition:
If entity1 is a single (i.e. simple, not composed) FT and the merge condition specifies a grouping of instances over that single FT, the instances are merged according to the following rule:
If no other aggregation function is provided:
i. *Identifiers*: A new Identifier is generated, following a defined augmentation rule (it still has to be clarified how augmentation rules are defined in OML – maybe as transformations on Entity2 objects).
ii. *Geometry*: Default Geometry is simply aggregated to a more complex type, e.g. from `LineString` to `MultipleLineString`. Of course, this will fail if the target schema doesn't allow this.
iii. *Strings*: Strings are just appended to each other, using the following pattern: `SourceID1::String;;SourceID2::String;...`
iv. *Classification codes*: Classification codes are ignored in merging. A separate attributive transformation can be used for this purpose.
v. *Other numeric attributes*: For these, average values are calculated.
Else if an aggregation function for the single FT is provided, merging is done according to that function.
The generated instance is added to *entitiy1Instances*
Else if entity1 is a composed entity (i.e. a conjunction of source FT)
Instances of entity1 are created that contain values for all attributes of all feature types, entity1 is composed of. Each created instance is added to *entity1Instances*.
If C contains a split condition:
Split entity1 instances according to the split condition and add each generated instance to entity1Instances

// **Actual mapping execution**
For each instance *Fs* in *entity1Instances*:
i. A Feature Ft of FeatureType FTt is created, with the identifier and the default geometry automatically copied from Ft if they have compatible attribute types;
ii. OML cells with attributive transformations are executed using Fs and Ft as specified;

EVALUATION AND SUMMARY

The method described in this paper was implemented into a Conceptual Schema Transformation Service (CST). This CST service can be used through multiple interfaces, such as a Java API and a Web Processing Service (WPS 1.0.0). In all cases, the service expects three inputs – the OML mapping, the source data as GML and the target GML application schema. For testing purposes, we encoded several matching tables into OML documents using a visual mapping editor (Reitz & Kuijper 2009). These matching tables included all of the mapping operations described by Letho (Letho 2007), such as reclassification, simple and spatial type conversions as well as some specific functions that deal with the creation of INSPIRE identifiers and geographical names. All of the OML documents could be executed successfully by the implemented service.

Performance

Since preprocessing is based only on the mapping and transformation functions are grouped, the complexity class of the approach as presented here should be O(n) to the number of source features

used, assuming that a feature store is available that makes use of an already calculated index. An example for such a feature store is a Web Feature Service.

To evaluate performance in practice, a mapping was created based on a matching table created for INSPIRE transformation testing for hydrographic data from Vorarlberg in Austria. The data set to be transformed had a size of 57 MB (in GML) and contained almost 30.000 features. For testing greater loads, this data set was replicated several times, to up a size of up to 921 MB. For the actual CST execution, throughput was indeed nearly linear with the amount of processed features. Performance for smaller numbers of features is lower (normalized at 1.0 for a set of 500 features) because of initialization times, but quite stable between 4.000 and 20.000 Features, before gradually getting lower to a normalized value of 1.25 at 464.000 Features. Absolute performance in terms of features per second depends on a huge range of factors, including the complexity of the features, the complexity of the mappings and the hardware and software that the performance evaluation was executed on. Especially the first two points can easily lead to a 100-fold difference in throughput. However, in the hydrography example above, throughputs of up to 9.000 features per second were achieved on a normal notebook PC. This throughput was reduced to about 400 features per second when also including geometry transformations, such as buffering or topology reconstruction operations.

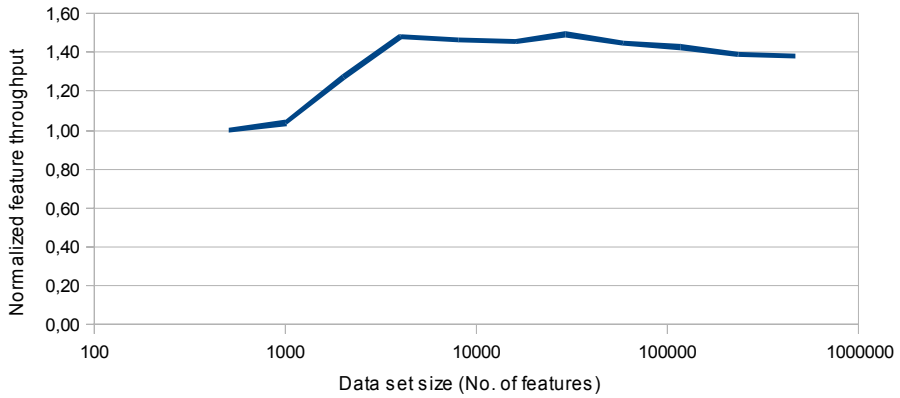

Figure 2: *Normalized throughput of Features of the prototypical CST implementation*

This was also due to the fact that the presented approach makes it possible to implement advanced querying and caching features. Because of the analysis of the cells in the mapping, the CST implementation can find out which Features need to be processed in an atomic transaction, can access the required source Features batch-wise, perform all necessary transformations and then continue with the next batch of Features.

Summary and Further Work

The ontology mapping language shows promise to offer high-expressivitiy for defining conceptual schema mappings in the geospatial domain. Within this paper, we have shown how to handle and employ the OML constructs when transforming spatial instance data. We provided a discussion on the appearance of the OML cells in the presence of different feature types, attribute and instance cardinalities. Further, we provided algorithms for executing mappings specified in a set of OML cells and discussed performance issues.

In future work, especially the handling of very large data sets access through standard interfaces such as WFS can be optimized, and the tools for creating and maintaining OML can be improved.

The prototypical implementation of the described algorithms is available under an open source license and can be downloaded freely[1].

ACKNOWLEDGEMENTS

The work described in this paper was partially funded within the HUMBOLDT Project (Contract SIP5-CT-2006-030962). The implementation of the CST Service was supported by Ana Belen Anton, Marian de Vries, Jan Jezek and Simon Templer. We would also like to thank the state Vorarlberg for providing datasets for the HUMBOLDT Project and the work described in this publication.

BIBLIOGRAPHY

Cruz, I.F. et al., 2007. A visual tool for ontology alignment to enable geospatial interoperability. *J. Vis. Lang. Comput.*, 18(3), 230-254.

Fan, L. & Xiao, T., 2009. An Automatic Method for Ontology Mapping. In *Knowledge-Based Intelligent Information and Engineering Systems*. pp. 661-669. Available at: http://dx.doi.org/10.1007/978-3-540-74829-8_81.

Letho, L., 2007. Schema Translation in a Web Service Based SDI. In *Proceedings of the 10th AGILE International Conference on Geographic Information Science*. 10th AGILE International Conference on Geographic Information Science 2007. Aalborg, Denmark.

Letho, L. & Sarjakovski, T., 2004. Schema Translations by XSLT for GML-Encoded Geospatial Data in Heterogeneous Web-Service Environment. In XXth ISPRS Congress. Instanbul.

Lutz, M. et al., 2009. Overcoming semantic heterogeneity in spatial data infrastructures. *Computers & Geosciences*, 35(4), 739-752.

Reitz, T. & Kuijper, A., 2009. Applying Instance Visualisation and Conceptual Schema Mapping for Geodata Harmonisation. In *Advances in GIScience*. pp. 173-194. Available at: http://dx.doi.org/10.1007/978-3-642-00318-9_9.

Scharffe, F., Euzenat, J. & Fensel, D., 2008. Towards design patterns for ontology alignment. In Fortaleza, Ceara, Brazil: ACM, pp. 2321-2325. Available at: http://portal.acm.org/citation.cfm?id=1364236.

Zhao, T. et al., 2008. Ontology-Based Geospatial Data Query and Integration. In *Geographic Information Science*. pp. 370-392. Available at: http://dx.doi.org/10.1007/978-3-540-87473-7_24.

[1] Documentation, source and builds are available from http://community.esdi-humboldt.eu .